Android Mobile App Development
8 Lab activities

Lab 1: GUI
Lab 2: Android Activity and User Interaction
Lab 3: Gesture and Multi-touch
Lab 4: Multiple Activity, URI, Linkify
Lab 5: Android services (Music Player) & Listview (AC Program Manager App)
Lab 6: Broadcast and Content Providers
Lab 7: Multiple Thread, Network and Internet
Lab 8: SQ Lite Database

LAB 1: Graphic User Interface (GUI)

User Interface and Layout in Android
Lab01: "Welcome to CS102"

Objectives:

The preparatory work for this lab has already been completed by us to save time, including **Android studio 2.1** & JDK installation. Students will learn how to create, build, run, and debug the first application in Android Studio.

In this lab, students will develop a simple layout (screen) including an ImageView (display college logo), three TextViews (display strings), and a TextClock showing current time. Furthermore, upon completion, students also learn:

Design Graphic User Interface (GUI):	**Java coding:**
- **Layout**: Relativelayout - **Background**: use color, xml background, or image - **Visual Controls**: TextView (display strings/text), ImageView (display image), TextClock (display current time) - **Set TextView/ImageView properties**: text, background, style, size, fontFamily - **App style & theme, and "app_name"** - **XML View properties**: padding, margin, gravity, match_parent, wrap_content, ...	- **Add Option Menu** including 3 items: "Help", "About", and "Exit"; - **Use Toast class** to pop up a message on screen; - **Use AlertDialog class** to add an **Alert Dialog** to ask user confirm "Exit" operation; **Best practices:** - Create ***color variables*** in **colors.xml file** - Create ***string variables*** in **strings.xml file** - Create ***background drawable*** in **/drawable/ folder** - Create ***menu's items*** in **/menu/ folder**

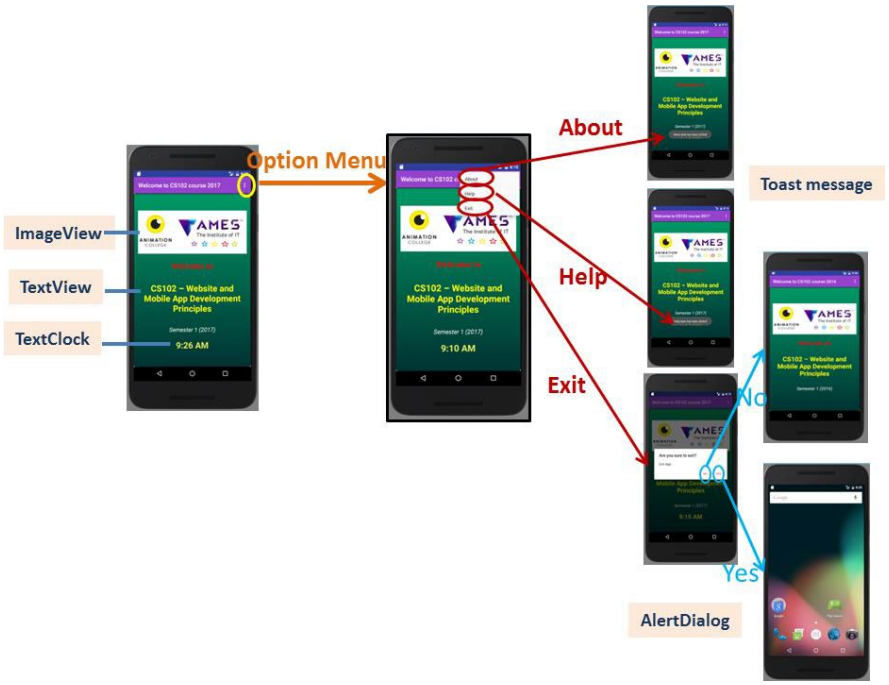


Figure: Lab01 wireframe

Layout in Android

Source: https://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a **layout** in two ways:

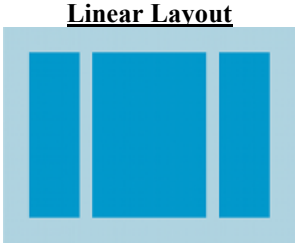
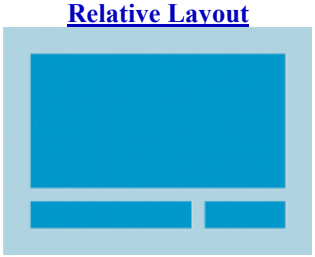
- **Declare UI elements in XML: Android** provides a straightforward XML vocabulary that corresponds to the **View classes** and subclasses, such as those for **widgets** and **layouts**. The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behaviour. Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile. Additionally, declaring the layout in XML makes it easier to visualize the structure of your UI, so it's easier to debug problems.
- **Instantiate layout elements at runtime**: Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Layout parameters: XML layout attributes named layout_*something* define layout parameters for the View that are appropriate for the ViewGroup in which it resides. All view groups include a width and height (layout_width and layout_height), and each view is required to define them. Many LayoutParams also include optional margins and borders. You can specify width and height with exact measurements, though you probably won't want to do this often. More often, you will use one of these constants to set the width or height:

- ***wrap_content*** tells your view to size itself to the dimensions required by its content.
- ***match_parent*** tells your view to become as big as its parent view group will allow.

In general, specifying a layout width and height using absolute units such as pixels is **not recommended**. Instead, using **relative measurements** such as density-independent pixel units (***dp***), ***wrap_content***, or ***match_parent***, is a better approach, because it helps ensure that your application will display properly across a variety of device screen sizes.

Common Layouts: Below are some of the more common layout types that are built into the Android platform.

Linear Layout	Relative Layout	Web View
		
A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.	Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).	Displays web pages.

"View" class: UI control elements or widgets

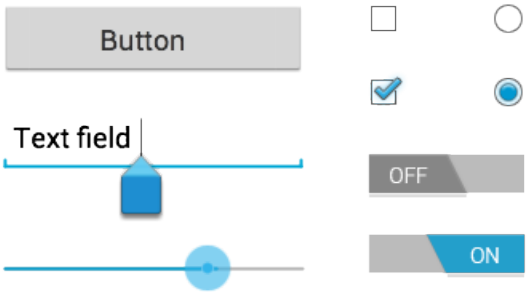
Source: https://www.tutorialspoint.com/android/android_user_interface_controls.htm

This class represents the **basic building block for user interface components**. A View 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 (**Button, TextView, ImageView, TextClock**, etc.).

Once you have created Views, there are typically a few types of common operations you may wish to perform:

- **Set properties:** for example setting the text of a TextView. The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.
- **Set focus:** The framework will handle moving focus in response to user input. To force focus to a specific view, call requestFocus().
- **Set up listeners:** Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. You can register such a listener using setOnFocusChangeListener(android.view.View.OnFocusChangeListener). Other view subclasses offer more specialized listeners. For example, a Button exposes a listener to notify clients when the button is clicked.
- **Set visibility:** You can hide or show views using setVisibility(int).

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as **buttons**, **text fields**, **seek bars**, **check box**, **zoom buttons**, **toggle buttons**, and many more.



UI ELEMENTS

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

Sr.No.	UI Control & Description
1	**TextView**: This control is used to display text to the user.
2	**EditText**: EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	**AutoCompleteTextView**: The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	**Button**: A push-button that can be pressed, or clicked, by the user to perform an action.
5	**ImageButton**: An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	**CheckBox**: An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	**ToggleButton**: An on/off button with a light indicator.
8	**RadioButton**: The RadioButton has two states: either checked or unchecked.
9	**RadioGroup**: A RadioGroup is used to group together one or more RadioButtons.
10	**ProgressBar**: The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	**Spinner**: A drop-down list that allows users to select one value from a set.
12	**TimePicker**: The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	**DatePicker**: The DatePicker view enables users to select a date of the day.

TextView control:

Source: https://www.tutorialspoint.com/android/android_textview_control.htm

*A **TextView** displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.*

TextView Attributes

Following are the important attributes related to **TextView control**. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Sr.No.	Attribute & Description
1	**android:id:** This is the ID which uniquely identifies the control.
2	**android:capitalize:** If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types. Don't automatically capitalize anything - 0 Capitalize the first word of each sentence - 1 Capitalize the first letter of every word - 2

	Capitalize every character - 3
3	**android:cursorVisible:** Makes the cursor visible (the default) or invisible. Default is false.
4	**android:editable:** If set to true, specifies that this TextView has an input method.
5	**android:fontFamily:** Font family (named by string) for the text.
6	**android:gravity:** Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
7	**android:hint:** Hint text to display when the text is empty.
8	**android:inputType:** The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
9	**android:maxHeight:** Makes the TextView be at most this many pixels tall.
10	**android:maxWidth:** Makes the TextView be at most this many pixels wide.
11	**android:minHeight:** Makes the TextView be at least this many pixels tall.
12	**android:minWidth:** Makes the TextView be at least this many pixels wide.
13	**android:password:** Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "true" or "false".
14	**android:phoneNumber:** If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".
15	**android:text:** Text to display.
16	**android:textAllCaps:** Present the text in ALL CAPS. Possible value either "true" or "false".
17	**android:textColor:** Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
18	**android:textColorHighlight:** Color of the text selection highlight.
19	**android:textColorHint:** Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
20	**android:textIsSelectable:** Indicates that the content of a non-editable text can be selected. Possible value either "true" or "false".
21	**android:textSize:** Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).
22	**android:textStyle:** Style (bold, italic, bolditalic) for the text. You can use or more of the following values separated by '|'. normal - 0 bold - 1 italic - 2
23	**android:typeface:** Typeface (normal, sans, serif, monospace) for the text. You can use or more of the following values separated by '|'. normal - 0 sans - 1 serif - 2 monospace - 3

References:

+ UI design: https://www.tutorialspoint.com/android/android_ui_design.htm :
+ Themes and styles: https://www.tutorialspoint.com/android/android_styles_and_themes.htm
+ Layout: https://www.tutorialspoint.com/android/android_user_interface_layouts.htm
+ UI controls: https://www.tutorialspoint.com/android/android_user_interface_controls.htm
+ UI pattern: https://www.tutorialspoint.com/android/android_ui_patterns.htm
+ Android - Alert Dialog Tutorial: https://www.tutorialspoint.com/android/android_alert_dialoges.htm

Mobile App Development Process

Step 1: Create a new Android Project
+ Target devices
+ Create an Activity & associated layout (screen)

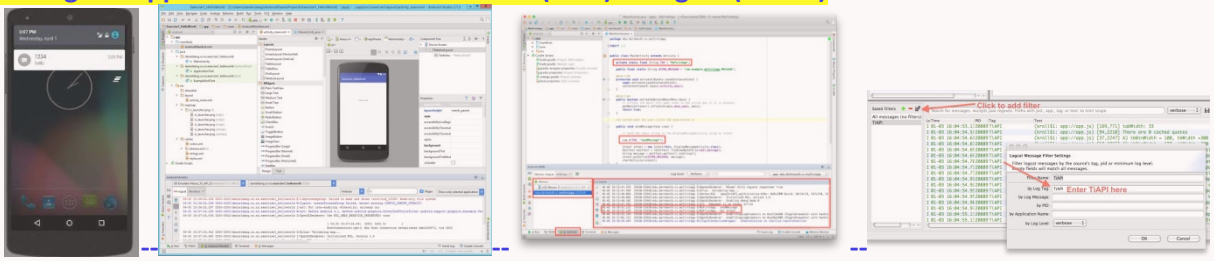
Step 2: Create Android Resources
+ res/drawables: images, shapes, animations,
+ res/values/colors.xml: define a list of colors for your app
+ res/values/strings.xml, res/values/dimens.xml, etc.

Step 3: Design graphic UI (res/layout) – XML layout
+ Design layout frame (nested layout)
+ Add Widget controls (TextView, Button, EditView, etc.)

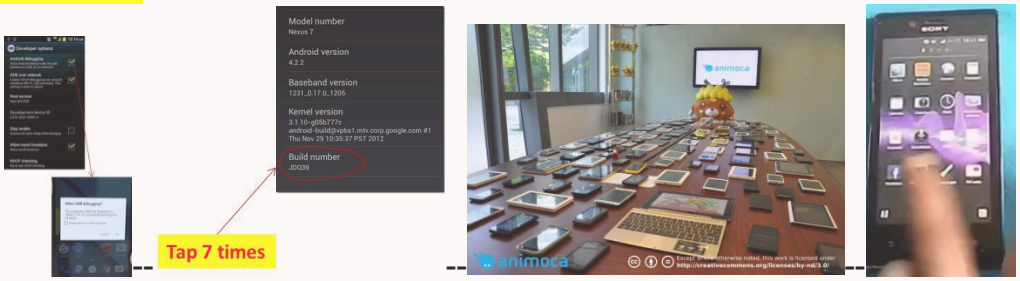
Step 4: Java coding – Activities and other java classes
+ Import classes, interfaces, and pacakges; Declare global objects, variables (right after class declaration)
+ Edit **onCreate() method** to create your Activity & its associated screen), to do casting/referencing (findViewID)
+ Edit **onClick() method** to respond to user events (clicks);

Step 5: Update AndroidManifest
+ App version, SDK target
+ Orientation
+ Permissions

Step 6: Debug the app with Android Virtual Devices (AVD) & Logcat (DDMS)



Step 7: Test on real devices



Step 8: Upload the app onto Google App Store

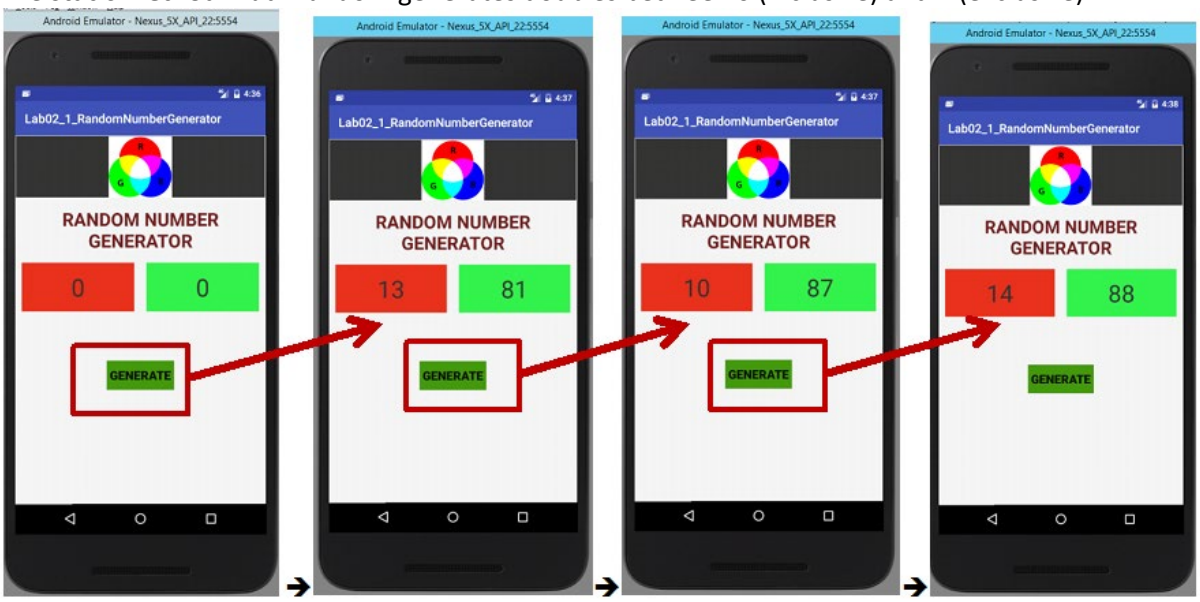
LAB 2: Android Activity and User Interaction

Activity and User Interaction in Android
Lab 02_1: Random Number Generator

1. Objectives

In this exercise, you will learn how to generate random number using Java class: **Random**. There are two principal means of generating random (really pseudo-random) numbers:

- The **Random class** generates random **integers, doubles, longs** and so on, in various ranges.
- The **static method** *Math.random* generates doubles between 0 (inclusive) and 1 (exclusive).



To generate random integers:

- Do not use *Math.random* (it produces doubles, not integers);
- Use the *Random class* to generate random integers between 0 and N.
- To generate a series of random numbers as a unit, you need to use a **single Random object** - do not create a new Random object for each new random number.

In this lab, students learn:

Design Graphic User Interface (GUI): - **ScrollView, RelativeLayout, LinearLayout** - Design **nested layout** -Visual controls: **Button, TextView, ImageView**	**Java coding: User Interaction** - Find id (widget control) on View: **findViewbyID()** - Set Event listener and response to that - Implement your class from "**onClickListener" interface** - 2D-array: `private int[][] operandRange = {{10,20}, {80,90}};`

Read more:

Investigate the **class Random** at: https://developer.android.com/reference/java/util/Random.html
Investigate the **class String** at: https://developer.android.com/reference/java/lang/String.html

Activity in Android:

Source: https://www.tutorialspoint.com/android/android_acitivities.htm

*An activity represents a **single screen** with a **user interface** just like **window** or **frame of Java**. Android activity is the subclass of **ContextThemeWrapper class**.*

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method.

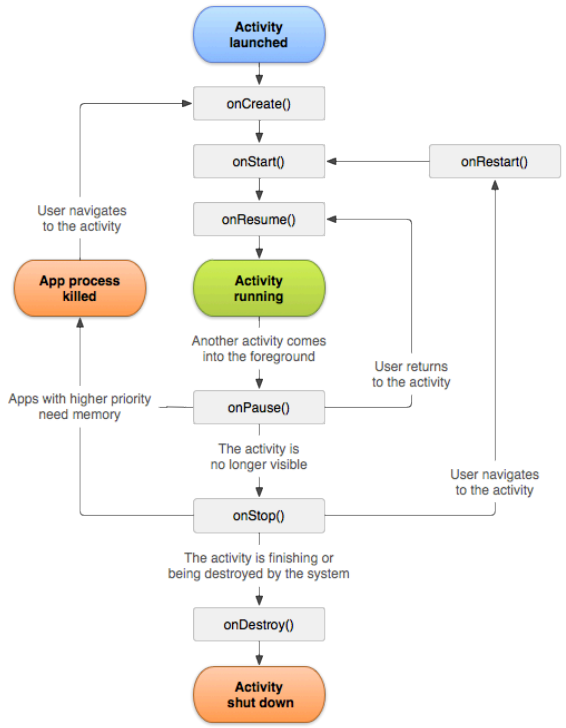
There are **two methods almost all subclasses of Activity** will implement:

- onCreate(Bundle) is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using findViewById(int) to retrieve the widgets in that UI that you need to interact with programmatically.
- onPause() is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the ContentProvider holding the data).

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, and are also central to how a user navigates within an app (as with the Back button) or between apps (as with the Recents button).

There is a sequence of **callback methods** that start up an activity and a sequence of callback methods that tear down an activity as shown in the beside Activity life cycle diagram

The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

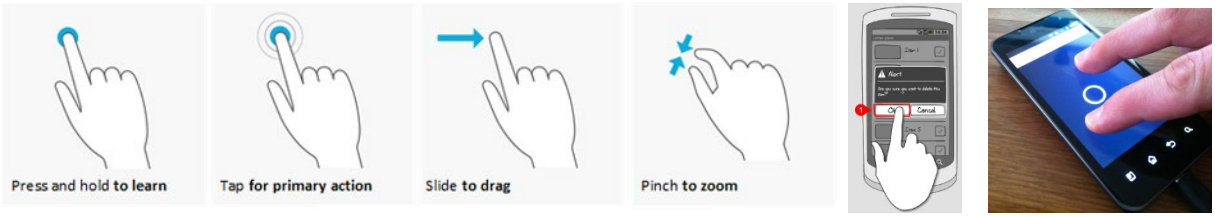


Activity life cycle diagram

Sr.No	Callback & Description
1	**onCreate():** This is the first callback and called when the activity is first created.
2	**onStart():** This callback is called when the activity becomes visible to the user.
3	**onResume():** This is called when the user starts interacting with the application.
4	**onPause():** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop():** This callback is called when the activity is no longer visible.
6	**onDestroy():** This callback is called before the activity is destroyed by the system.
7	**onRestart():** This callback is called when the activity restarts after stopping it.

User Interaction Handling

Source: https://www.tutorialspoint.com/android/android_event_handling.htm



User events: press, click, touch, tap, slide, pinch, etc.

Events are a useful way to collect data about a user's interaction with interactive components of Applications. Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

There are following three concepts related to Android Event Management:

- **Event Listeners** − An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.
- **Event Listeners Registration** − Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.
- **Event Handlers** − When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

Event Listeners & Event Handlers:

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	**onCreateContextMenuItemListener()** This is called when the context menu is being built(as the result of a sustained "long click)

Button control

A Button is a Push-button which can be pressed, or clicked, by the user to perform an action.

Button Attributes

Following are the important attributes related to Button control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class:

Sr.No	Attribute & Description
1	**android:autoText:** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom:** This is the drawable to be drawn below the text.
3	**android:drawableRight:** This is the drawable to be drawn to the right of the text.
4	**android:editable:** If set, specifies that this TextView has an input method.
5	**android:text:** This is the Text to display.

Inherited from **android.view.View** Class:

Attribute	Description
1	**android:background:** This is a drawable to use as the background.
2	**android:contentDescription:** This defines text that briefly describes content of the view.
3	**android:id:** This supplies an identifier name for this view.
4	**android:onClick:** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility:** This controls the initial visibility of the view.

Random number

Source: https://developer.android.com/reference/java/util/Random.html

An instance of this class is used to generate a stream of pseudorandom numbers. The class uses a 48-bit seed, which is modified using a linear congruential formula.

If two instances of Random are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers. In order to guarantee this property, particular algorithms are specified for the class Random. Java implementations must use all the algorithms shown here for the class Random, for the sake of absolute portability of Java code. However, subclasses of class Random are permitted to use other algorithms, so long as they adhere to the general contracts for all the methods.

The algorithms implemented by class Random use a protected utility method that on each invocation can supply up to 32 pseudorandomly generated bits.

Many applications will find the method random() simpler to use.

Instances of java.util.Random are threadsafe. However, the concurrent use of the same java.util.Random instance across threads may encounter contention and consequent poor performance. Consider instead using ThreadLocalRandom in multithreaded designs.

Instances of java.util.Random are not cryptographically secure. Consider instead using SecureRandom to get a cryptographically secure pseudo-random number generator for use by security-sensitive applications.

Activity and User Interaction in Android
Lab02_2: Quiz App in Android

1. Objectives

This exercise shows you how to create a simple quiz app in Android.

- The app reads questions from **an array** defined **in a string-array stored in res/values/strings.xml file**.
- The questions are asked one by one. When user clicks to select the correct answer, a **Toast Message** pops up to indicate whether the answer is right or wrong.
- User clicks the "**Next question" botton** to proceed.

Figure 1: AndroidQuizApp wireframe

In this lab, students learn:

Design Graphic User Interface (GUI): **- Linearlayout** **- Visual controls:** TextView, ImageView, Button **- Resources:** String-array defined in /res/values/strings.xml file	**Java coding:** **- 1D-array in java** **- *Conditional* and "*If …else …*" checking condition** **- Data types: String, Float, etc.**

References:

+ Arrays.xml: https://developer.android.com/samples/MediaRouter/res/values/arrays.html
+ Strings.xml: https://developer.android.com/samples/MediaRouter/res/values/strings.html
+ Java array: https://www.tutorialspoint.com/java/java_arrays.htm

Android Resources Organizing & Accessing for your project

Source: https://www.tutorialspoint.com/android/android_resources.htm

There are many more items which you use to build a good Android application. Apart from coding (in java) for the application, you take care of **various other resources** like **static content** that your code uses, such as **bitmaps**, **colors**, **layout definitions**, **user interface strings**, **animation instructions**, and more. These resources are always maintained separately in various sub-directories under **res/ directory** of the project.

Organize sources in Android Studio:

Directory	Resource Type
anim/	XML files that define property animations. They are saved in **res/anim/ folder** and accessed from the **R.anim** class.
color/	XML files that define a state list of colors. They are saved in **res/color/** and accessed from the **R.color** class.
drawable/	Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in **res/drawable/** and accessed from the **R.drawable** class.
layout/	XML files that define a user interface layout. They are saved in **res/layout/** and accessed from the **R.layout** class.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in **res/menu/** and accessed from the **R.menu** class.
raw/	Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the resource ID, which is *R.raw.filename* to open such raw files.
values/	XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory: • **arrays.xml** for resource arrays, and accessed from the **R.array** class. • **integers.xml** for resource integers, and accessed from the **R.integer** class. • **bools.xml** for resource boolean, and accessed from the **R.bool** class. • **colors.xml** for color values, and accessed from the **R.color** class. • **dimens.xml** for dimension values, and accessed from the **R.dimen** class. • **strings.xml** for string values, and accessed from the **R.string** class. • **styles.xml** for styles, and accessed from the **R.style** class.
xml/	Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time.

Arrays

Source: https://www.tutorialspoint.com/java/java_arrays.htm

Java provides a data structure, the **array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables.

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable:

Syntax:

```
dataType[] arrayRefVar;   // preferred way.
Or:
dataType arrayRefVar[];  // works but not preferred way.
```

Note: *The style **dataType[] arrayRefVar** is preferred. The style **dataType arrayRefVar[]** comes from the C/C++ language and was adopted in Java to accommodate C/C++ programmers.*

Creating Arrays

You can create an array by using the new operator with the following syntax:

```
arrayRefVar = new dataType[arraySize];
```

The above statement does two things:

- It creates an array using new dataType[arraySize].
- It assigns the reference of the newly created array to the variable arrayRefVar.

Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement, as shown below:

```
dataType[] arrayRefVar = new dataType[arraySize];
```

Alternatively you can create arrays as follows:

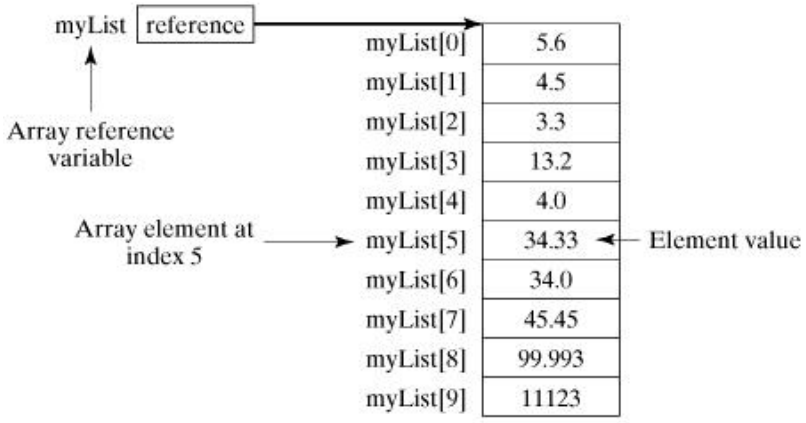
```
dataType[] arrayRefVar = {value0, value1, ..., valuek};
```

The array elements are accessed through the **index**. Array indices are 0-based; that is, they start from 0 to **arrayRefVar.length-1**.

Example: Following statement declares an array variable, myList, creates an array of 10 elements of double type and assigns its reference to myList:

```
double[] myList = new double[10];
```

Following picture represents array myList. Here, myList holds ten double values and the indices are from 0 to 9.



Processing Arrays

When processing array elements, we often use either **for** loop or **foreach** loop because all of the elements in an array are of the same type and the size of the array is known.

Example: Here is a complete example showing how to create, initialize, and process arrays:

```
public class TestArray {
  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};
    // Print all the array elements
    for (int i = 0; i < myList.length; i++) {
      System.out.println(myList[i] + " ");
    }
    // Summing all elements
    double total = 0;
    for (int i = 0; i < myList.length; i++) {
      total += myList[i];
```

```
    }
    System.out.println("Total is " + total);
    // Finding the largest element
    double max = myList[0];
    for (int i = 1; i < myList.length; i++) {
      if (myList[i] > max) max = myList[i];
    }
    System.out.println("Max is " + max);
  }
}
```

The foreach Loops

JDK 1.5 introduced a new for loop known as foreach loop or enhanced for loop, which enables you to traverse the complete array sequentially without using an index variable.
The following code displays all the elements in the array myList:

```
public class TestArray {

  public static void main(String[] args) {
    double[] myList = {1.9, 2.9, 3.4, 3.5};

    // Print all the array elements
    for (double element: myList) {
      System.out.println(element);
    }
  }
}
```

Passing Arrays to Methods

Just as you can pass primitive type values to methods, you can also pass arrays to methods. For example, the following method displays the elements in an **int** array:

```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

You can invoke it by passing an array. For example, the following statement invokes the printArray method to display 3, 1, 2, 6, 4, and 2:

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Returning an Array from a Method

A method may also return an array. For example, the following method returns an array that is the reversal of another array:

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];

  for (int i = 0, j = result.length - 1; i < list.length; i++, j--) {
    result[j] = list[i];
  }
  return result;
}
```

The Arrays Class

The java.util.Arrays class contains various static methods for sorting and searching arrays, comparing arrays, and filling array elements. These methods are overloaded for all primitive types.

Sr.No.	Method & Description
1	**public static int binarySearch(Object[] a, Object key)** Searches the specified array of Object (Byte, Int , double, etc.) for the specified value using the binary search algorithm. The array must be sorted prior to making this call. This returns index of the search key, if it is contained in the list; otherwise, it returns (– (insertion point + 1)).
2	**public static boolean equals(long[] a, long[] a2)** Returns true if the two specified arrays of longs are equal to one another. Two arrays are considered equal if both arrays contain the same number of elements, and all corresponding pairs of elements in the two arrays are equal. This returns true if the two arrays are equal. Same method could be used by all other primitive data types (Byte, short, Int, etc.)
3	**public static void fill(int[] a, int val)** Assigns the specified int value to each element of the specified array of ints. The same method could be used by all other primitive data types (Byte, short, Int, etc.)
4	**public static void sort(Object[] a)** Sorts the specified array of objects into an ascending order, according to the natural ordering of its elements. The same method could be used by all other primitive data types (Byte, short, Int, etc.)

Strings Class

Source: https://www.tutorialspoint.com/java/java_strings.htm

Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects.
The Java platform provides the String class to create and manipulate strings.

Creating Strings

The most direct way to create a string is to write:

```
String greeting = "Hello world!";
```

Whenever it encounters a string literal in your code, the compiler creates a String object with its value in this case, "Hello world!'.
As with any other object, you can create String objects by using the new keyword and a constructor. The String class has 11 constructors that allow you to provide the initial value of the string using different sources, such as an array of characters. **Example:**

```
public class StringDemo {
  public static void main(String args[]) {
    char[] helloArray = { 'h', 'e', 'l', 'l', 'o', '.' };
    String helloString = new String(helloArray);
    System.out.println( helloString );
  }
}
```

Note: *The String class is immutable, so that once it is created a String object cannot be changed. If there is a necessity to make a lot of modifications to Strings of characters, then you should use String Buffer & String Builder Classes.*

String Length

Methods used to obtain information about an object are known as **accessor methods**. One accessor method that you can use with strings is the length() method, which returns the number of characters contained in the string object. The following program is an example of **length()**, method String class.

```
public class StringDemo {
  public static void main(String args[]) {
    String palindrome = "Dot saw I was Tod";
    int len = palindrome.length();
    System.out.println( "String Length is : " + len );
  }
}
```

Concatenating Strings

The String class includes a method for concatenating two strings:

```
string1.concat(string2);
```

This returns a new string that is string1 with string2 added to it at the end. You can also use the concat() method with string literals, as in:

```
"My name is ".concat("Zara");
```

Strings are more commonly concatenated with the + operator, as in:

```
"Hello," + " world" + "!"
```

which results in –

```
"Hello, world!"
```

String Methods

Here is the list of methods supported by String class:

Method & Description
char charAt(int index): Returns the character at the specified index.
int indexOf(int ch): Returns the index within this string of the first occurrence of the specified character.
int length(): Returns the length of this string.
String replaceFirst(String regex, String replacement): Replaces the first substring of this string that matches the given regular expression with the given replacement.
String[] split(String regex): Splits this string around matches of the given regular expression.
String[] split(String regex, int limit): Splits this string around matches of the given regular expression.
String substring(int beginIndex): Returns a new string that is a substring of this string.
String substring(int beginIndex, int endIndex): Returns a new string that is a substring of this string.
char[] toCharArray(): Converts this string to a new character array.
String toLowerCase(): Converts all of the characters in this String to lower case using the rules of the default locale.
String toLowerCase(Locale locale): Converts all of the characters in this String to lower case using the rules of the given Locale.
String toString(): This object (which is already a string!) is itself returned.
String toUpperCase(): Converts all of the characters in this String to upper case using the rules of the default locale.
String toUpperCase(Locale locale): Converts all of the characters in this String to upper case using the rules of the given Locale.
String trim(): Returns a copy of the string, with leading and trailing whitespace omitted.
static String valueOf(primitive data type x): Returns the string representation of the passed data type argument.
Integer.parseInt(String str): This method is used to get the primitive data type of a certain String. parseXxx() is a static method and can have one argument or two. **parseInt(String s)** – returns an integer (decimal only).
string1.contains(String string2): returns true if and only if the "string1" contains the "string2".

EXTRA WORK (OPTIONAL)

Activity and User Interaction in Android

Lab02_3: Calculator App (OPTIONAL)

This exercise shows you how to create a simple calculator application in Android. It will have the following functionality: division, multiplication, subtraction, and addition.

Figure 1: Calculator Wireframe

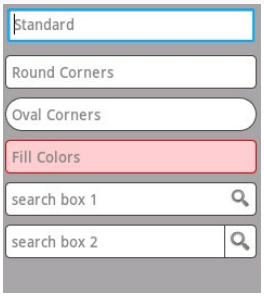
In this lab, students learn:

Design Graphic User Interface (GUI): - **Linearlayout, TableLayout** - Analyse and design **nested layout** - Change **app launcher icon** - **Button, EditView, TextView, ImageView** - Develop resources for reuse: **Font, Text, Color**	**Java coding: User Interaction** - Find id (widget control) on View: **findViewbyID()** - Set Event listener and response to that - Implement your class from "**onClickListener" interface** - Data types: String, float, Integer. - Conditional: switch ... case, don't forget "default" - Format decimal number

Android - EditText Control

Source: https://www.tutorialspoint.com/android/android_edittext_control.htm

A EditText is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities.



STYLES OF EDIT TEXT

EditText Attributes

Following are the important attributes related to EditText control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class:

Sr.No	Attribute & Description
1	**android:autoText:** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom:** This is the drawable to be drawn below the text.
3	**android:drawableRight:** This is the drawable to be drawn to the right of the text.
4	**android:editable:** If set, specifies that this TextView has an input method.
5	**android:text:** This is the Text to display.

Inherited from **android.view.View** Class:

Sr.No	Attribute & Description
1	**android:background:** This is a drawable to use as the background.
2	**android:contentDescription:** This defines text that briefly describes content of the view.
3	**android:id:** This supplies an identifier name for this view.
4	**android:onClick:** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility:** This controls the initial visibility of the view.

References:
+ View.OnClickListner interface: https://developer.android.com/reference/android/view/View.OnClickListener.html
+ Event handling: https://www.tutorialspoint.com/android/android_event_handling.htm
+ Formatter: https://developer.android.com/reference/java/util/Formatter.html
+ Input Events: https://developer.android.com/guide/topics/ui/ui-events.html
+ User interaction handling with Android: https://code.tutsplus.com/tutorials/android-sdk-user-interaction--mobile-20342

You develop the Version 2 of calculator app using TableLayout and all the Buttons instead of EditText for entering the inputs (number 1, number 2) as following:

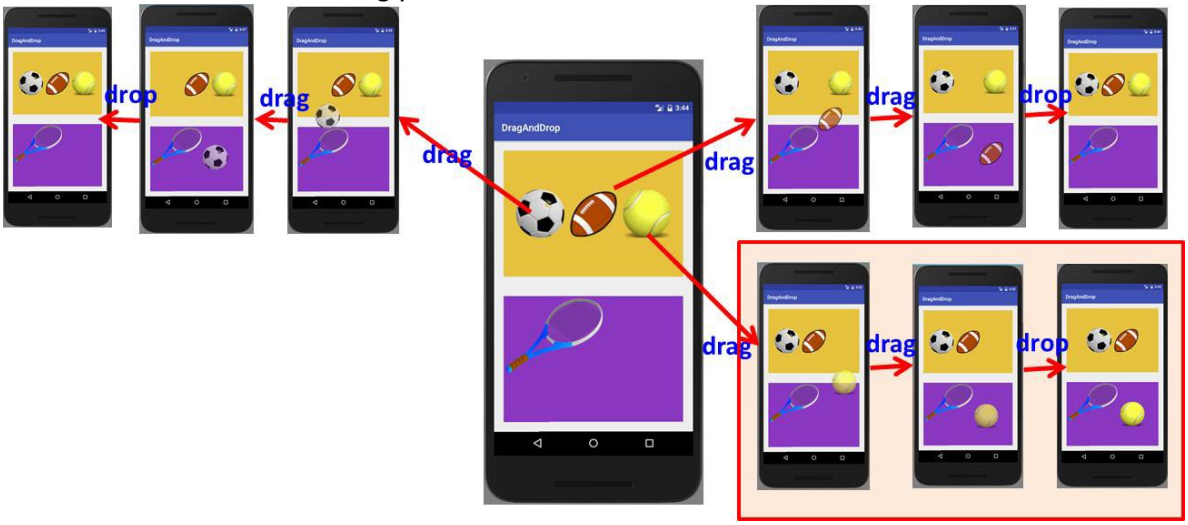
LAB 3: Gesture and multi-touch

"Drag and Drop", Multiple touch, and GridView in Android
Lab03_1: Select right object

Objective

Android's Drag and drop framework enables you to drag and drop data between views. This tutorial shows you **how to drag an image from one layout and drop it into another layout.**

- You'll need Android 3.0, API Level 11 (Honeycomb).
- The drag starts with a **gesture**, like when the **user touches a view**. A **drag shadow** is then created to represent the view that we want to drag.
- All the while, the system sends out **drag events** containing information about the dragged object. This includes information about its **location and the state of the drag operation** (whether it has started, ended, etc.).
- We register listeners to **listen for the gesture** that starts the drag and also to **listen for the drag events**. We then handle the events accordingly.



This tutorial app **displays three balls** at the **top of the screen** and **a tennis racket at the bottom**.

- The main layout contains **two LinearLayout layouts**, a top one **shown in dark yellow** and a bottom one **shown in violet**. The top layout contains three image views and the bottom layout one image view
- The user **must drag and drop the correct ball to the bottom of the screen**. Dragging the ball sets the original image view to invisible. What we see is the Drag Shadow, a copy of the original image.
- Once dropped, **only the tennis ball will remain at the bottom of the screen**. The other two balls will **bounce back to the top**.

References:

[1]. Android- Drag and Drop: https://www.tutorialspoint.com/android/android_drag_and_drop.htm
[2]. Drag and Drop: https://developer.android.com/guide/topics/ui/drag-drop.html
[3]. Drap and Drop tutorial: http://www.101apps.co.za/index.php/articles/drag-and-drop-tutorial.html
[4]. Android - Using drag and drop in your application – Tutorial: http://www.vogella.com/tutorials/AndroidDragAndDrop/article.html
[5]. Android SDK: Implementing Drag-and-Drop Functionality: https://code.tutsplus.com/tutorials/android-sdk-implementing-drag-and-drop-functionality--mobile-14402
[6]. Android Drag and Drop: http://codingjunkie.net/android-drag-and-drop-part1/

Drag-and-Drop in Android

Android drag/drop framework allows the users to move data from one View to another View in the current layout using a graphical drag and drop gesture. As of **API 11** drag and drop of view onto other views or view groups is supported.The framework includes following three important components to support drag & drop functionality −

- **Drag event class.**
- **Drag listeners.**
- **Helper methods and classes.**

The Drag/Drop Process

There are basically **four steps or states** in the drag and drop process:

- **Started** − This event occurs when you start dragging an item in a layout, your application calls ***startDrag()* method** to tell the system to start a drag. The arguments inside **startDrag() method** provide the data to be dragged, metadata for this data, and a callback for drawing the drag shadow.
  The system first responds by calling back to your application to get a drag shadow. It then displays the drag shadow on the device.
  Next, the system sends a drag event with action type ***ACTION_DRAG_STARTED*** to the registered drag event listeners for all the View objects in the current layout.
  To continue to receive drag events, including a possible drop event, a drag event listener must return **true**, If the drag event listener returns false, then it will not receive drag events for the current operation until the system sends a drag event with action type **ACTION_DRAG_ENDED**.
- **Continuing** − The user continues the drag. System sends **ACTION_DRAG_ENTERED** action followed by **ACTION_DRAG_LOCATION** action to the registered drag event listener for the View where dragging point enters. The listener may choose to alter its View object's appearance in response to the event or can react by highlighting its View.
  The drag event listener receives a **ACTION_DRAG_EXITED** action after the user has moved the drag shadow outside the bounding box of the View.
- **Dropped** − The user releases the dragged item within the bounding box of a View. The system sends the View object's listener a drag event with action type **ACTION_DROP**.
- **Ended** − Just after the action type **ACTION_DROP**, the system sends out a drag event with action type **ACTION_DRAG_ENDED** to indicate that the drag operation is over.

The DragEvent Class

The **DragEvent** represents an event that is sent out by the system at various times during a drag and drop operation. This class provides few Constants and important methods which we use during Drag/Drop process.

Constants

Following are all constants integers available as a part of DragEvent class.

Sr.No.	Constants & Description
1	**ACTION_DRAG_STARTED:** Signals the start of a drag and drop operation.
2	**ACTION_DRAG_ENTERED:** Signals to a View that the drag point has entered the bounding box of the View.
3	**ACTION_DRAG_LOCATION:** Sent to a View after ACTION_DRAG_ENTERED if the drag shadow is still within the View object's bounding box.
4	**ACTION_DRAG_EXITED:** Signals that the user has moved the drag shadow outside the bounding box of the View.
5	**ACTION_DROP:** Signals to a View that the user has released the drag shadow, and the drag point is within the bounding box of the View.
6	**ACTION_DRAG_ENDED:** Signals to a View that the drag and drop operation has concluded.

Methods

Following are few important and most frequently used methods available as a part of DragEvent class.

Sr.No.	Constants & Description
1	**int getAction():** Inspect the action value of this event..
2	**ClipData getClipData():** Returns the ClipData object sent to the system as part of the call to startDrag().
3	**ClipDescription getClipDescription():** Returns the ClipDescription object contained in the ClipData.
4	**boolean getResult():** Returns an indication of the result of the drag and drop operation.
5	**float getX():** Gets the X coordinate of the drag point.
6	**float getY():** Gets the Y coordinate of the drag point.
7	**String toString():** Returns a string representation of this DragEvent object.

Listening for Drag Event

If you want any of your views within a Layout should respond Drag event then your view either implements **View.OnDragListener** or setup **onDragEvent(DragEvent)** callback method.

When the system calls the method or listener, it passes to them a **DragEvent object** explained above. You can have both a listener and a callback method for View object. If this occurs, the system first calls the listener and then defined callback as long as listener returns true.

The combination of the *onDragEvent(DragEvent)* method and *View.OnDragListener* is analogous to the combination of the **onTouchEvent()** and **View.OnTouchListener** used with touch events in old versions of Android.

Starting a Drag Event

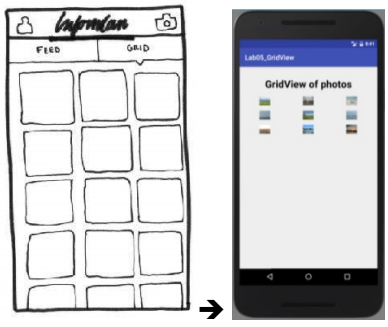
You start with creating a **ClipData** and **ClipData.Item** for the data being moved. As part of the *ClipData* object, supply metadata that is stored in a **ClipDescription** object within the ClipData. For a drag and drop operation that does not represent data movement, you may want to use **null** instead of an actual object.

Next either you can extend extend **View.DragShadowBuilder** to create a drag shadow for dragging the view or simply you can use *View.DragShadowBuilder(View)* to create a default drag shadow that's the same size as the View argument passed to it, with the touch point centered in the drag shadow.

"Drag and Drop", Multiple touch, and GridView in Android
Lab03_2: GridView of Photos

Objectives

In this tutorial, you will learn to implement a GridView of photo in 2D scrolling grid and then display the photo in full screen when user clicks on it.



GRID VIEW

Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter.** An adapter actually bridges between UI

components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.
The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

Following are the **important attributes specific to GridView**:

Sr.No	Attribute & Description
1	**android:id:** This is the ID which uniquely identifies the layout.
2	**android:columnWidth:** This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
3	**android:gravity:** Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	**android:horizontalSpacing:** Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
5	**android:numColumns:** Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.
6	**android:stretchMode** Defines how columns should stretch to fill the available empty space, if any. This must be either of the values − • none − Stretching is disabled. • spacingWidth − The spacing between each column is stretched. • columnWidth − Each column is stretched equally. • spacingWidthUniform − The spacing between each column is uniformly stretched..
7	**android:verticalSpacing:** Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.

References:
[1]. Android Grid View: https://www.tutorialspoint.com/android/android_grid_view.htm

Android - Multitouch

Source: https://www.tutorialspoint.com/android/android_multitouch.htm

Multi-touch gesture happens when more then one finger touches the screen at the same time. Android allows us to detect these gestures.
Android system generates the following touch events whenever multiple fingers touches the screen at the same time.

Sr.No	Event & description
1	**ACTION_DOWN:** For the first pointer that touches the screen. This starts the gesture.
2	**ACTION_POINTER_DOWN:** For extra pointers that enter the screen beyond the first.
3	**ACTION_MOVE:** A change has happened during a press gesture.
4	**ACTION_POINTER_UP:** Sent when a non-primary pointer goes up.
5	**ACTION_UP:** Sent when the last pointer leaves the screen.

So in order to detect any of the above mention event , you need to override **onTouchEvent()** method and check these events manually. Its syntax is given below −

```
public boolean onTouchEvent(MotionEvent ev){
  final int actionPeformed = ev.getAction();

  switch(actionPeformed){
    case MotionEvent.ACTION_DOWN:{
      break;
    }
```

```
    case MotionEvent.ACTION_MOVE:{
      break;
    }
    return true;
  }
}
```

In these cases, you can perform any calculation you like. For example zooming , shrinking e.t.c. In order to get the co-ordinates of the X and Y axis, you can call **getX()** and **getY()** method. Its syntax is given below −

```
final float x = ev.getX();
final float y = ev.getY();
```

Apart from these methods, there are other methods provided by this MotionEvent class for better dealing with multitouch. These methods are listed below −

Sr.No	Method & description
1	**getAction():** This method returns the kind of action being performed
2	**getPressure():** This method returns the current pressure of this event for the first index
3	**getRawX():** This method returns the original raw X coordinate of this event
4	**getRawY():** This method returns the original raw Y coordinate of this event
5	**getSize():** This method returns the size for the first pointer index
6	**getSource():** This method gets the source of the event
7	**getXPrecision():** This method return the precision of the X coordinates being reported
8	**getYPrecision():** This method return the precision of the Y coordinates being reported

Android - Gestures

Source: https://www.tutorialspoint.com/android/android_gestures.htm

Android provides special types of touch screen events such as **pinch, double tap, scrolls , long presses and flinch**. These are all known as gestures. Android provides GestureDetector class to receive motion events and tell us that these events correspond to gestures or not. To use it , you need to create an object of GestureDetector and then extend another class with **GestureDetector.SimpleOnGestureListener** to act as a listener and override some methods. Its syntax is given below:

```
GestureDetector myG;
myG = new GestureDetector(this,new Gesture());

class Gesture extends GestureDetector.SimpleOnGestureListener{
  public boolean onSingleTapUp(MotionEvent ev) {
  }
  public void onLongPress(MotionEvent ev) {
  }
  public boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX,
  float distanceY) {
  }
  public boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX,
  float velocityY) {
  }
}
```

Handling Pinch Gesture

Android provides **ScaleGestureDetector** class to handle gestures like pinch e.t.c. In order to use it, you need to instantiate an object of this class. Its syntax is as follow −

```
ScaleGestureDetector SGD;
```

```
SGD = new ScaleGestureDetector(this,new ScaleListener());
```

The first parameter is the context and the second parameter is the event listener. We have to define the event listener and override a function **OnTouchEvent** to make it working. Its syntax is given below –

```
public boolean onTouchEvent(MotionEvent ev) {
  SGD.onTouchEvent(ev);
  return true;
}

private class ScaleListener extends ScaleGestureDetector.SimpleOnScaleGestureListener {
  @Override
  public boolean onScale(ScaleGestureDetector detector) {
    float scale = detector.getScaleFactor();
    return true;
  }
}
```

Apart from the pinch gestures , there are other methods available that notify more about touch events. They are listed below:

Sr.No	Method & description
1	**getEventTime():** This method get the event time of the current event being processed..
2	**getFocusX():** This method get the X coordinate of the current gesture's focal point.
3	**getFocusY():** This method get the Y coordinate of the current gesture's focal point.
4	**getTimeDelta():** This method return the time difference in milliseconds between the previous accepted scaling event and the current scaling event.
5	**isInProgress():** This method returns true if a scale gesture is in progress..
6	**onTouchEvent(MotionEvent event):** This method accepts MotionEvents and dispatches events when appropriate.

Extra work (Optional):

GUI in Android: Spinner, Radio Buttons, EditText, and TextView

Lab 03_3: Temperature Converter

Objectives:

This app has an educational purpose only. It demonstrates an Android application to convert temperatures between **Celsius**, **Fahrenheit**, **Kelvin** and **Rankine** scales. Simply input one temperature (with or withou decimal point) and the other temperature is converted for you.

Celsius temperature scale: The Celsius scale is widely known as the centigrade scale because it is divided into 100 degrees. The **freezing point** is taken as 0 degrees Celsius and the **boiling point** as 100 degrees Celsius. It is named for the Swedish astronomer Anders Celsius, who established the scale in 1742.
Read more: https://en.wikipedia.org/wiki/Celsius

The Fahrenheit scale: is a thermodynamic temperature scale, where the **freezing point** of water is 32 degrees Fahrenheit (°F) and the **boiling point** 212°F (at standard atmospheric pressure). This puts the boiling and freezing points of water exactly 180 degrees apart. **Absolute zero** is defined as -459.67°F.
Read more: https://en.wikipedia.org/wiki/Fahrenheit

The Kelvin scale: Kelvin is a unit of measure for temperature based upon an absolute scale and and the experimental evidence that **absolute zero** is -273.15ºC. **Zero** on the Celsius scale (0°C) is now defined as the equivalent to 273.15K, with a temperature difference of 1 deg C equivalent to a difference of 1K, meaning the unit size in each

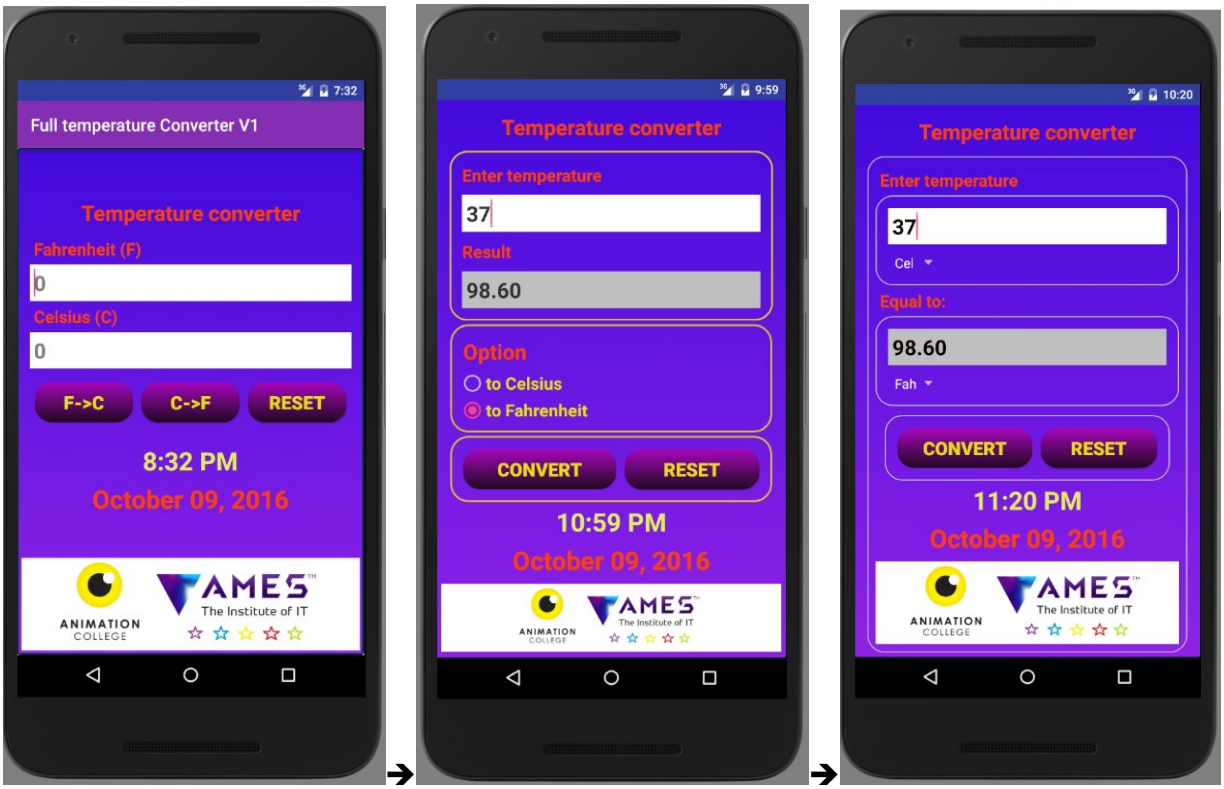
scale is the same. This means that 100°C, previously defined as the **boiling point of water**, is now defined as the equivalent to 373.15K.
Read more: https://en.wikipedia.org/wiki/Kelvin

Rankine scale: Rankine temperature scale has an **absolute zero**, below which temperatures do not exist, and using a degree of the **same size as that used by the Fahrenheit temperature scale**. The temperature scale is named after the Scottish engineer and physicist William John Macquorn Rankine, who proposed it in 1859.
Read more: https://en.wikipedia.org/wiki/Rankine_scale

Temperature conversion formulae:

Kelvin temperature conversion formulae		
	from kelvins	to kelvins
Celsius	[°C] = [K] − 273.15	[K] = [°C] + 273.15
Fahrenheit	[°F] = [K] × 9⁄5 − 459.67	[K] = ([°F] + 459.67) × 5⁄9
Rankine	[°R] = [K] × 9⁄5	[K] = [°R] × 5⁄9

Celsius temperature conversion formulae		
	from Celsius	to Celsius
Fahrenheit	[°F] = [°C] × 9⁄5 + 32	[°C] = ([°F] − 32) × 5⁄9
Kelvin	[K] = [°C] + 273.15	[°C] = [K] − 273.15
Rankine	[°R] = ([°C] + 273.15) × 9⁄5	[°C] = ([°R] − 491.67) × 5⁄9



Version 1.0

Version 1.0:
TextView, EditText, Buttons

Version 2.0

Version 2.0:
TextView, EditText, Buttons, RadioButtons

Version 3.0

Version 3.0:
TextView, EditText, Buttons, Spinner

Students will learn:

+ Design UI:
. Widget controls: **TextView, EditText, Button, RadioButton, Spinner, and TextClock;**
. Create **android resources** (strings/texts, colors, dimensions, styles, drawables, etc.): organize and access;

+ Java coding:
. How to **use these resources in xml file java file**;
. How to build **test cases to verify and test your app**;
. How to debug you app with **Android Virtual Devices (AVD) & Dalvik Debug Monitor Server (DDMS) and Logcat**

. What's **Exception**? How to **catch & handle Exception?**
. Edit AndroidManifest.xml: **codeversion, screen orientation, no title (fullscreen);**
. Learn to use **Radio button;**
. Learn to use **List & ArrayAdapter & Spinner**;
. Learn to create and use **methods (function)** in the same java file;
. Learn to **create a separate class (ConverterUtility) containing converting functions**;
. Learn to **define style (text color, text size) of spinner item using java code**;

+ Readings:

1. Android Resources Organizing & Accessing: https://www.tutorialspoint.com/android/android_resources.htm
2. Event handling: https://www.tutorialspoint.com/android/android_event_handling.htm
3. Java – Exceptions: https://www.tutorialspoint.com/java/java_exceptions.htm
4. DDMS: Logcat & Logging: https://developer.android.com/studio/profile/ddms.html
5. Logcat Command-line tool: https://developer.android.com/studio/command-line/logcat.html

Development process

Step 1: Create a new Android Project
+ Target devices
+ Create an Activity & associated layout (screen)

Step 2: Create Android Resources
+ res/drawables: images, shapes, animations,
+ res/values/colors.xml: define a list of colors for your app
+ res/values/strings.xml:
+ res/values/dimens.xml:

Step 3: Design graphic UI (res/layout)
+ Design layout frame (nested layout)
+ Add Widget controls (TextView, Button, EditView, etc.)

Step 4: Java coding
+ Import classes, interfaces, and pacakges;
+ Declare global objects, variables (right after class declaration)
+ Edit **onCreate() method** to create your Activity & its associated screen), to do casting/referencing (findViewID) to layout elements, to set listener to detect user interaction/events (click, touch, tap, etc.) on elments;
+ Edit **onClick() method** to respond to user events (clicks);

Step 5: Update AndroidManifest
+ App version, SDK target
+ Orientation

Debugging you app with Android Virtual Devices (AVD) & Logcat (DDMS)

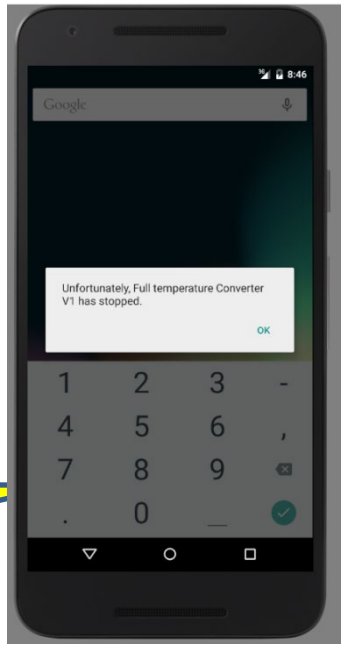
Define test cases for your application

A test case, in software engineering, is a **set of conditions** under which a tester will determine whether an application, software system or one of its features is working as it was originally established for it to do. The mechanism for determining whether a software program or system has passed or failed such a test is known as a **test oracle**. In some settings, **an oracle could be a requirement or use case**, while in others it could be a **heuristic**.

It may take **many test cases** to determine that a software program or system is considered sufficiently scrutinized to be released.

+ Test cases for the temperature converter:

Test cases	Input values	Output values	Expected outputs
1	-40°F	?	-40°C
2	0°F	?	-17.78°C
3	32°F	?	0°C
4	-40°C	?	-40°F
5	0°C	?	32°F
6	100°C	?	212°F
7	"Empty"	?	Crash



Use Dalvik Debug Monitor Server (DDMS) and Logcat

+ Dalvik Debug Monitor Server (DDMS):

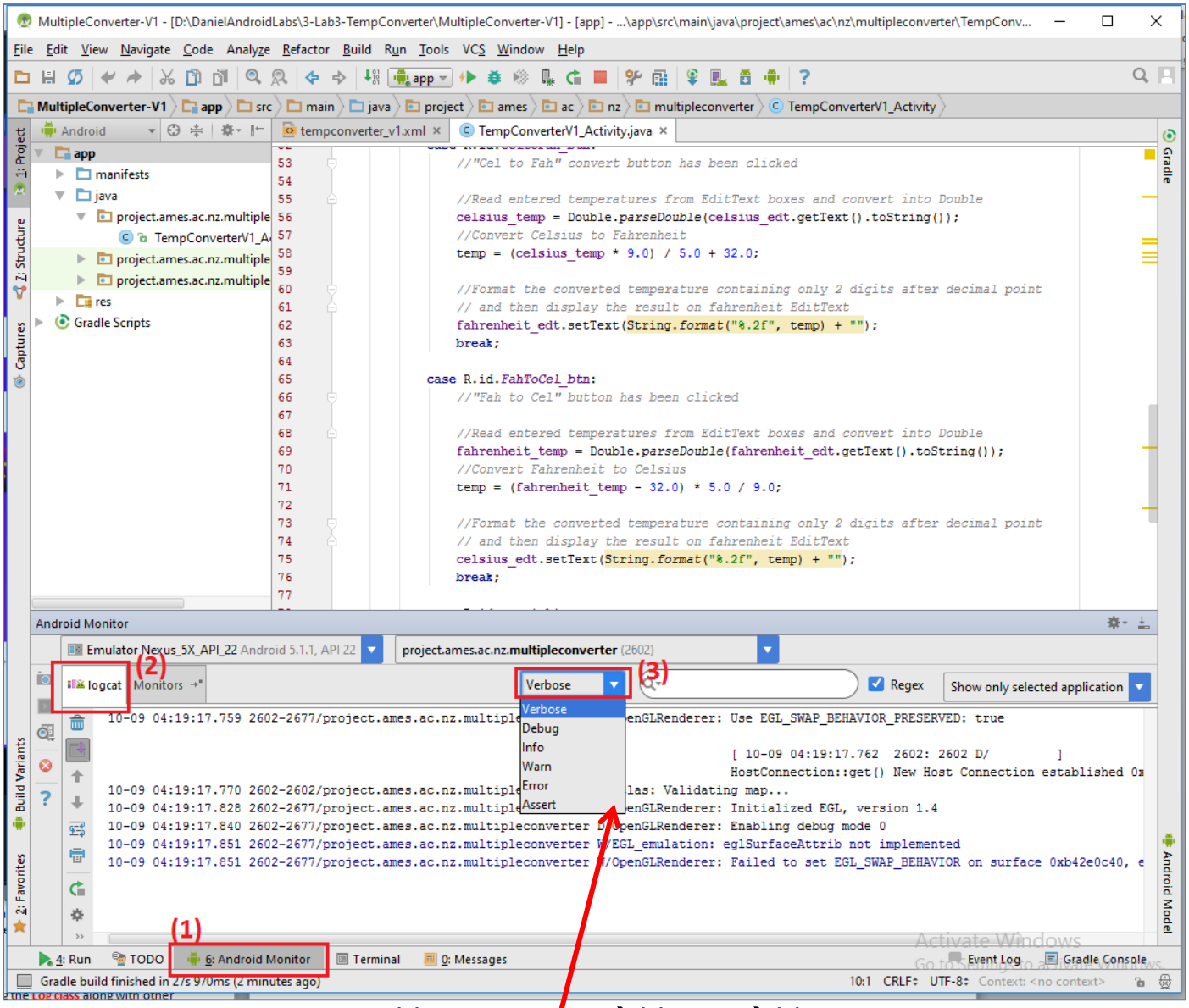
Android Studio includes a **debugging tool called the Dalvik Debug Monitor Server (DDMS),** which provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more.
For more information on accessing logcat from DDMS, instead of the command line, see Using DDMS at:
http://android.magicer.xyz/tools/debugging/ddms.html

The **Android logging system** provides a mechanism for **collecting and viewing system debug output**. Logs from various applications and portions of the system are collected in a series of circular buffers, **which then can be viewed and filtered by the logcat command.** You can use logcat from an ADB shell to view the log messages.

+ Using Logcat:

LogCat is integrated into DDMS, and outputs the messages that you print out using the **Log class** along with other system messages such as stack traces when exceptions are thrown.
In order to view Logcat window, follow the steps:

(1) Click **"6. Android Monitor"** at the bottom bar of the Android Studio IDE;
(2) Choose "**logcat**" tab;
(3) Select filter "Verbose"



Logcat monitor: (1) Android Monitor → (2) Logcat → (3) Filter: Verbose

Log is a logging class that you can utilize in your code to print out messages to the LogCat. Common logging methods include:

- `v(String, String)` (verbose)
- `d(String, String)` (debug)
- `i(String, String)` (information)
- `w(String, String)` (warning)
- `e(String, String)` (error)

Logging messages on LogCat

For example:

```
Log.i("MyActivity", "Get item number " + position);
```

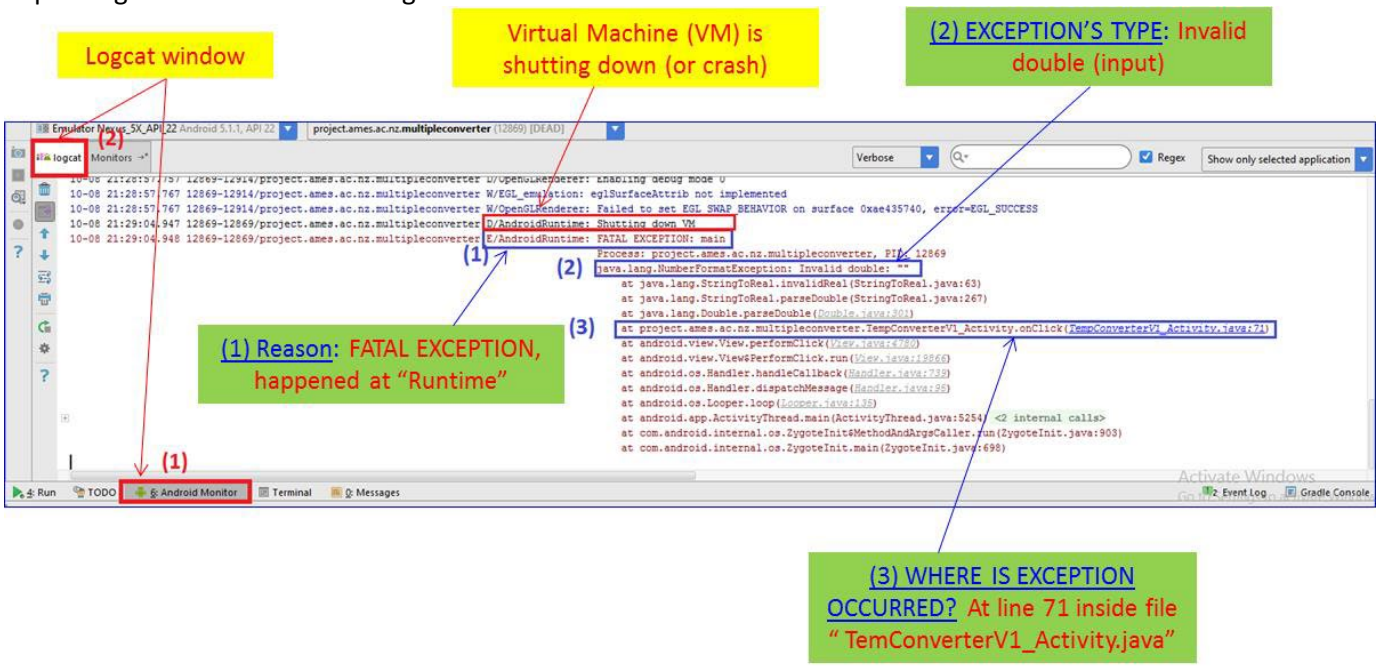
The LogCat will then output something like:

```
I/MyActivity(1557): Get item number 1
```

Why crash? What's Exception? How to catch & handle Exception

Why? Crash due to Exceptions

+ Open Logcat window and investigate Error:



Logcat window: show the error why the application is crashed.

What's Exception?

An **exception** (or **exceptional event**) is a problem that arises during the execution of a program. When an **Exception** occurs, the normal flow of the program is disrupted and the program/Application **terminates abnormally**, which is not recommended, therefore, **these exceptions are to be handled**.

An exception can occur for many different reasons. Following are some scenarios where an exception occurs:

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- A network connection has been lost in the middle of communications or the JVM has run out of memory.

Here it is. The reason why your app crashes

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these, we have **three categories of Exceptions**. You need to understand them to know how exception handling works in Java.

1. **Checked exceptions** − A checked exception is an exception that occurs at the compile time, these are also called as **compile time exceptions**. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of (handle) these exceptions. For example, if you use FileReader class in your program to read data from a file, if the file specified in its constructor doesn't exist, then a FileNotFoundException occurs, and the compiler prompts the programmer to handle the exception.
2. **Unchecked exceptions** − An unchecked exception is an exception that occurs at the time of execution. These are also called as Runtime Exceptions. These include programming bugs, such as logic errors or improper use of an API. Runtime exceptions are ignored at the time of compilation. For example, if you have declared an array of size 5 in your program, and trying to call the 6th element of the array then an ArrayIndexOutOfBoundsExceptionexception occurs.
3. **Exception Hierarchy** - All exception classes are subtypes of the java.lang.Exception class. The exception class is a subclass of the Throwable class. Other than the exception class there is another subclass called Error which is derived from the Throwable class. Errors are abnormal conditions that happen in case of severe failures, these are not handled by the Java programs. Errors are generated to indicate errors generated by the runtime environment. Example: JVM is out of memory. Normally, programs cannot recover from errors.

How to catch "Exceptions" to avoid CRASH? – TRY … CATCH

A method **catches an exception** using a combination of the **try** and **catch keywords**. A **try/catch block** is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

Syntax:

```
try {
     //Try block:
     //Protect codes that possibly generate an exception
}
catch (ExceptionName e) {
   //Catch block:
   //statements to handle any exceptions
}
```

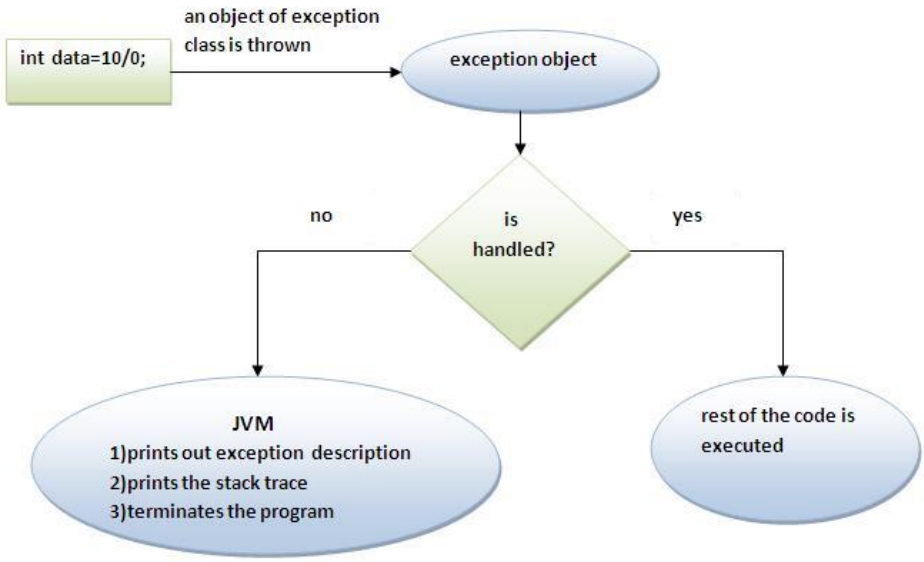
The code which is prone to exceptions is placed in the try block. When an exception occurs, that exception occurred is handled by catch block associated with it. Every try block should be immediately followed either by a catch block or finally block.

A catch statement involves declaring the type of exception you are trying to catch. If an exception occurs in protected code, the **catch block (or blocks)** that follows the try is checked. If the type of exception that occurred is listed in a catch block, the exception is passed to the catch block much as an argument is passed into a method parameter.

Internal working of java try-catch block: The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

- Prints out exception description.
- Prints the stack trace (Hierarchy of methods where the exception occurred).
- Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.



Internal working of java try-catch block: http://www.javatpoint.com/try-catch-block

Reading more:

[1]. Java – Exceptions: https://www.tutorialspoint.com/java/java_exceptions.htm

Add “version code” to AndroidManifest.xml

+ Open AndroidManifest.xml and add your code version:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="project.ames.ac.nz.multipleconverter"
    android:versionCode="1"
    android:versionName="1.0">
```

Specify SDK version (minimum level of the Android API)

+ Define the SDK version for your app when compiling:

```
<!--Define SDK version when compiling your app-->
<uses-sdk
    android:minSdkVersion="19"
    android:targetSdkVersion="24" />
```

Define “Screen” orientation

+ The screen orientation attribute is provided by the activity element in the Android Manifest.Xml file. The orientations provided by the activity are **Portrait**, **Landscape**, **Sensor**, **Unspecified** and so on. To perform a screen orientation activity you define the properties in the Android Manifest.Xml file.

+ Test you app in two modes:

- “**landscape” orientation**
- “**portrait” orientation**

As can be seen, your app is designed only for “portrait” orientation because the “landscape” mode distords the view. In order to define your app only displayed in “portrait” mode, add the following attribute:

```
<activity android:name=".TempConverterV1_Activity"
    android:screenOrientation="portrait">
```

Here is the complete androidmanisfet.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="project.ames.ac.nz.multipleconverter"
    android:versionCode="1"
    android:versionName="1.0">

    <!--Define SDK version when compiling your app-->
    <uses-sdk
        android:minSdkVersion="19"
        android:targetSdkVersion="24" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">

        <activity android:name=".TempConverterV1_Activity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
```

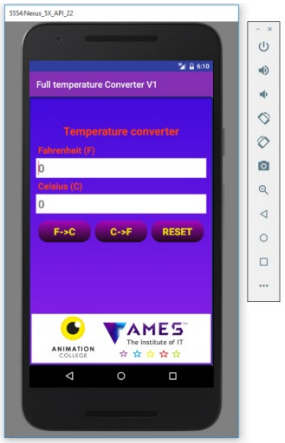
Set your “launch icon”

Set your “app_name”

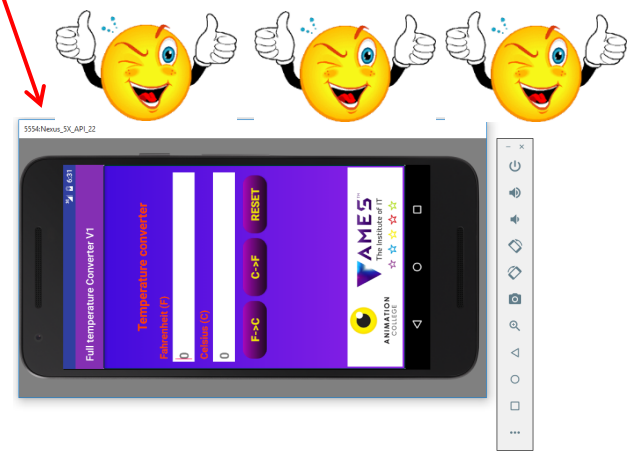
Delare activity “TempConverterV1_Activity

Activity “Calculator” will be launched first when your app is opened

+ Test again the screen orientation mode:



“Portrait” orientation



“landscape” orientation

LAB 4: Multiple Activity, URI, Linkify

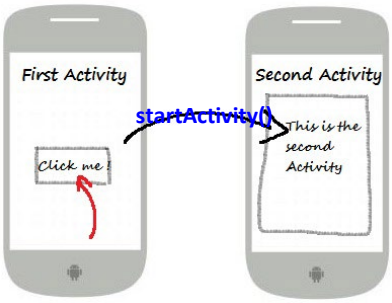
Multiple Activity, Unifrom Resource Identifier (URI) and Linkify
Lab 04_1: Open Internet website using URI and Linkify

Figure 1: Wireframes for our **ImpliciteIntent application**

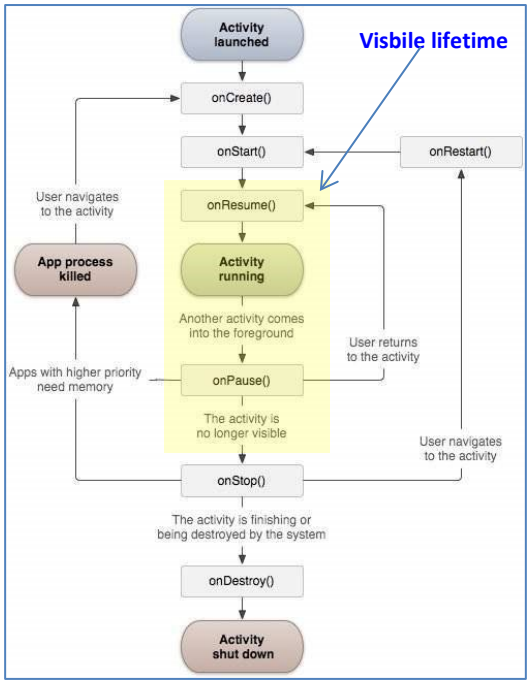
Objectives:

This exercise shows you how to work with multiple Activity and How to exchange data between those Activity.

Activity:



- **An Activity** is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map. Each activity is given a window in which to draw its user interface.
- **Life cycle: Android system initiates its program** with in an Activity starting with a call on **onCreate() callback** method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity as shown in the below Activity life cycle diagram:



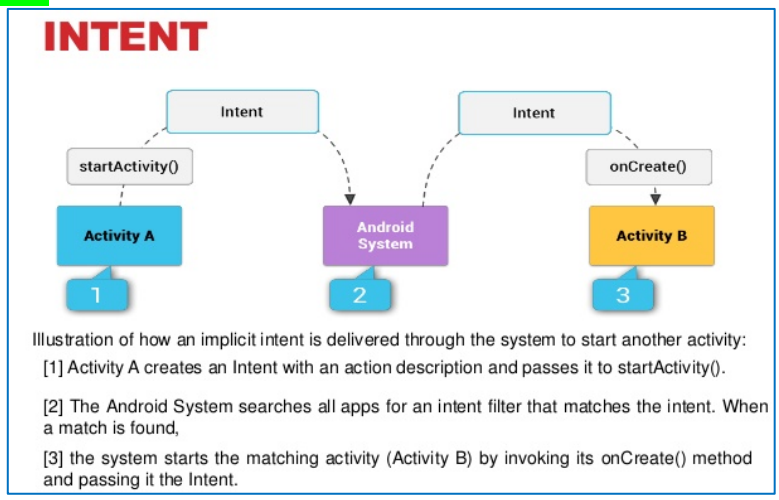
Callback	Description
onCreate()	This is the first callback and called when the activity is first created.
onStart()	This callback is called when the activity becomes visible to the user.
onResume()	This is called when the user starts interacting with the application.
onPause()	The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
onStop()	This callback is called when the activity is no longer visible.
onDestroy()	This callback is called before the activity is destroyed by the system.
onRestart()	This callback is called when the activity restarts after stopping it.

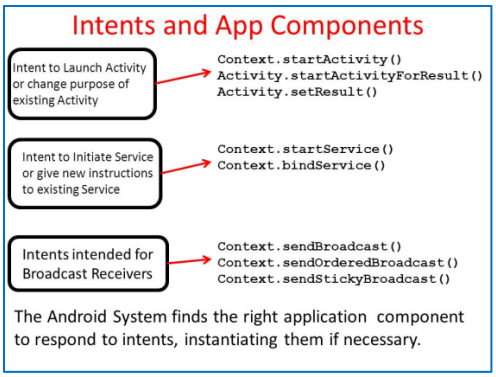
When we develop an app, we usually deal with several Activity (or associated screens) in Android. In Android, **Intents are the objects** for sending and receiving data between Android Activity.

Read more:

1. Activities: https://developer.android.com/guide/components/activities.html
2. Android Activity: https://www.tutorialspoint.com/android/android_acitivities.htm

Intents





An Intent is a **messaging object** you can use to **request an action** from another app component. Although intents facilitate communication between **components** in several ways, there are three fundamental use-cases:

To start an activity:

- **An Activity represents a single screen in an app**. You can start a new instance of an Activity by passing an **Intent** to startActivity(). The **Intent** describes the activity to start and carries any necessary data.
- If you want to receive a result from the activity when it finishes, call startActivityForResult(). Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback. For more information, see the Activities guide.

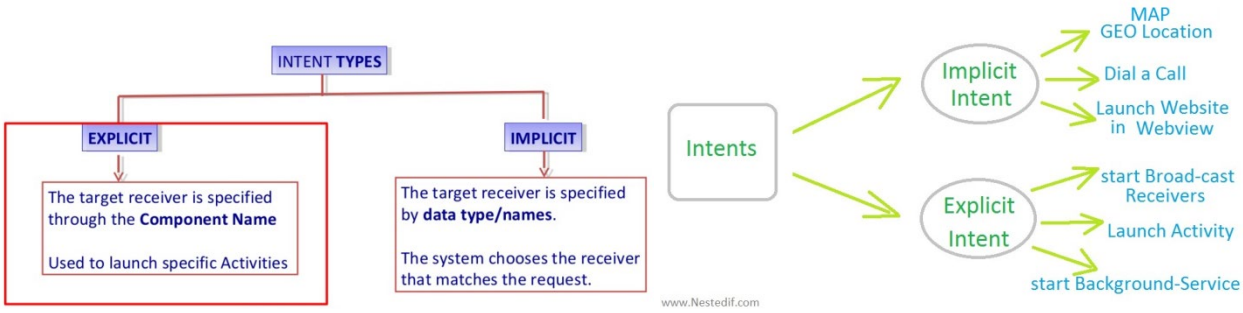
To start a service:

- **A Service is a component that performs operations in the background without a user interface**. You can start a service to perform a one-time operation (such as download a file) by passing an Intent to startService(). The Intent describes the service to start and carries any necessary data.

- If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService(). For more information, see the Services guide.

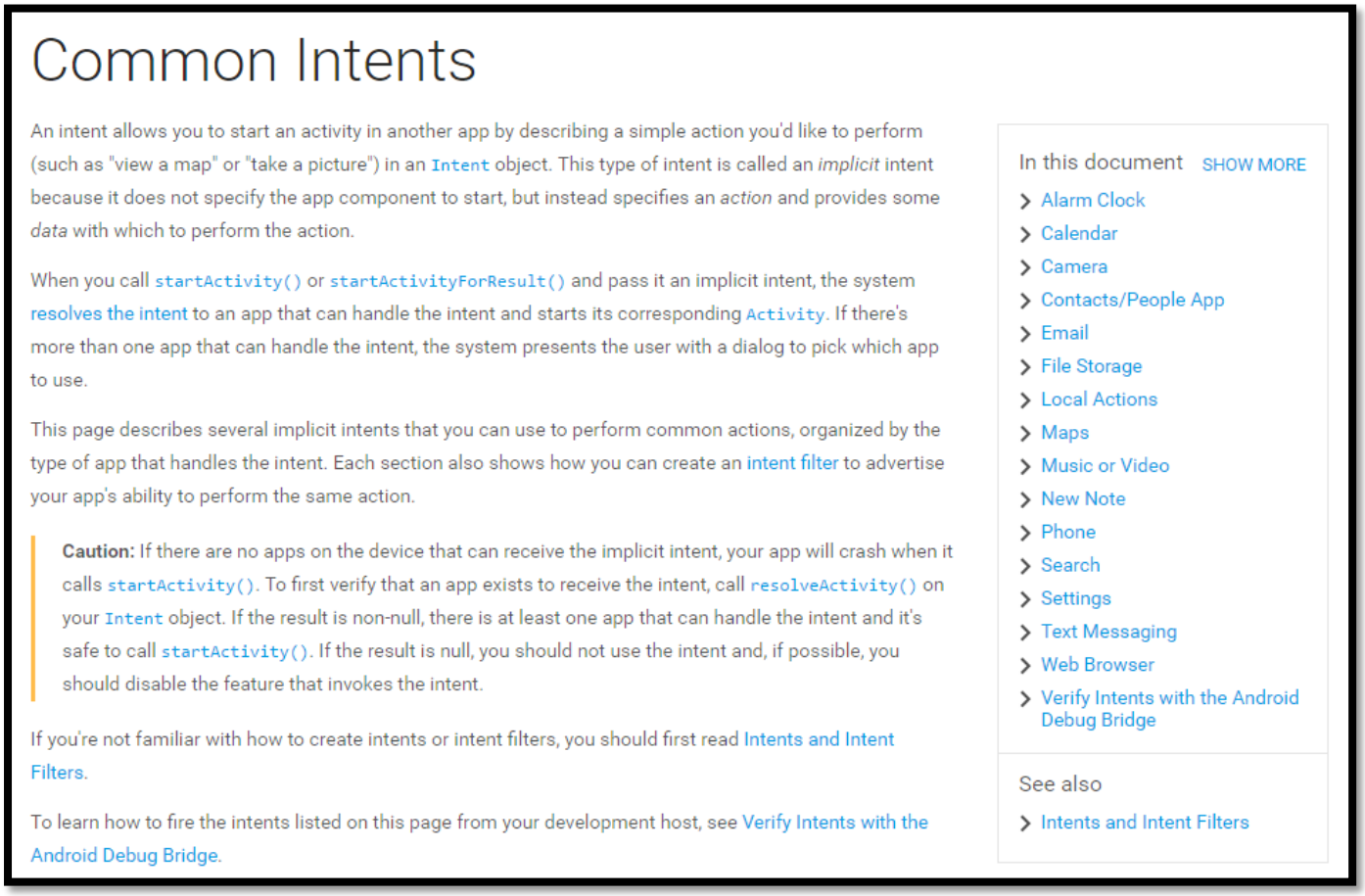
To deliver a broadcast:

- **A broadcast is a message that any app can receive**. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast()



There are two types of intents:

- **Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.



Common Intents

An intent allows you to start an activity in another app by describing a simple action you'd like to perform (such as "view a map" or "take a picture") in an `Intent` object. This type of intent is called an *implicit* intent because it does not specify the app component to start, but instead specifies an *action* and provides some *data* with which to perform the action.

When you call `startActivity()` or `startActivityForResult()` and pass it an implicit intent, the system resolves the intent to an app that can handle the intent and starts its corresponding `Activity`. If there's more than one app that can handle the intent, the system presents the user with a dialog to pick which app to use.

This page describes several implicit intents that you can use to perform common actions, organized by the type of app that handles the intent. Each section also shows how you can create an intent filter to advertise your app's ability to perform the same action.

> **Caution:** If there are no apps on the device that can receive the implicit intent, your app will crash when it calls `startActivity()`. To first verify that an app exists to receive the intent, call `resolveActivity()` on your `Intent` object. If the result is non-null, there is at least one app that can handle the intent and it's safe to call `startActivity()`. If the result is null, you should not use the intent and, if possible, you should disable the feature that invokes the intent.

If you're not familiar with how to create intents or intent filters, you should first read Intents and Intent Filters.

To learn how to fire the intents listed on this page from your development host, see Verify Intents with the Android Debug Bridge.

In this document SHOW MORE

- Alarm Clock
- Calendar
- Camera
- Contacts/People App
- Email
- File Storage
- Local Actions
- Maps
- Music or Video
- New Note
- Phone
- Search
- Settings
- Text Messaging
- Web Browser
- Verify Intents with the Android Debug Bridge

See also

- Intents and Intent Filters

+ Read more:

1. Intents and Intent Filters: https://developer.android.com/guide/components/intents-filters.html
2. Intents and Filters: https://www.tutorialspoint.com/android/android_intents_filters.htm

Unifrom Resource Identifier (URI) and Linkify

Unifrom Resource Identifier (URI)

A **Uniform Resource Identifier (URI) object** is usually used to tell a ContentProvider what we want to access by reference. It is an immutable one-to-one mapping to a resource or data.

The **method Uri.parse** creates a new Uri object from a properly format String.

URI are characterized by the following definitions:

- **Uniform**: Uniformity provides several benefits: it allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ; it allows uniform semantic interpretation of common syntactic conventions across different types of resource identifiers; it allows introduction of new types of resource identifiers without interfering with the way that existing identifiers are used; and, it allows the identifiers to be reused in many different contexts, thus permitting new applications or protocols to leverage a pre-existing, large, and widely-used set of resource identifiers.
- **Resource**: A resource can be anything that has identity. Familiar examples include an electronic document, an image, a service (e.g., "today's weather report for Los Angeles"), and a collection of other resources. Not all resources are network "retrievable"; e.g., human beings, corporations, and bound books in a library can also be considered resources. The resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content---the entities to which it currently corresponds---changes over time, provided that the conceptual mapping is not changed in the process.
- **Identifier**: An identifier is an object that can act as a reference to something that has identity. In the case of URI, the object is a sequence of characters with a restricted syntax.

Example URI: The following examples illustrate URI that are in common use.

- ftp://ftp.is.co.za/rfc/rfc1808.txt -- ftp scheme for File Transfer Protocol services
- gopher://spinaltap.micro.umn.edu/00/Weather/California/Los%20Angeles -- gopher scheme for Gopher and Gopher+ Protocol services
- http://www.math.uio.no/faq/compression-faq/part1.html -- http scheme for Hypertext Transfer Protocol services
- mailto:mduerst@ifi.unizh.ch -- mailto scheme for electronic mail addresses
- news:comp.infosystems.www.servers.unix -- news scheme for USENET news groups and articles
- telnet://melvyl.ucop.edu/ -- telnet scheme for interactive services via the TELNET Protocol

Linkify

public class Linkify
extends Object
java.lang.Object
↳ android.text.util.Linkify

Linkify take a piece of text and a regular expression and turns all of the regex matches in the text into clickable links. This is particularly useful for matching things like email addresses, web URLs, etc. and making them actionable. Alone with the pattern that is to be matched, a URL scheme prefix is also required. Any pattern match that does not begin with the supplied scheme will have the scheme prepended to the matched text when the clickable URL is created. For instance, if you are matching web URLs you would supply the scheme http://.
If the pattern matches example.com, which does not have a URL scheme prefix, the supplied scheme will be prepended to create http://example.com when the clickable URL link is created.

Multiple Activity, Unifrom Resource Identifier (URI) and Linkify
Lab04_2: Online Application Form App

Objectives

Develop "Online Application Form" that allow users to fill the application form and then send it out to the AC college by Email or SMS.

The application form includes the following information:

ApplicationForm

Name: ..... (***Required***)
Email: ..... (**Required**)
Date of Birth (DD/MM/YYYY): .....
Contact Number/Mobile: .....
Programme you're interested in (dropdown list):

- Bachelor of Animation (BA),
- Bachelor of Creative Software (BCS),
- Diploma in Computing and Network Support
- .....

Previous tertiary study: (Yes/No)
Submit buttons: (1) Send Email, (2) Send SMS

Required: the app checks **name field** and **email field** to make sure that they are filled in. If not, pop up a message to ask user to enter User **Name** and User **Email**.

Students have to test the app on the real Android device.

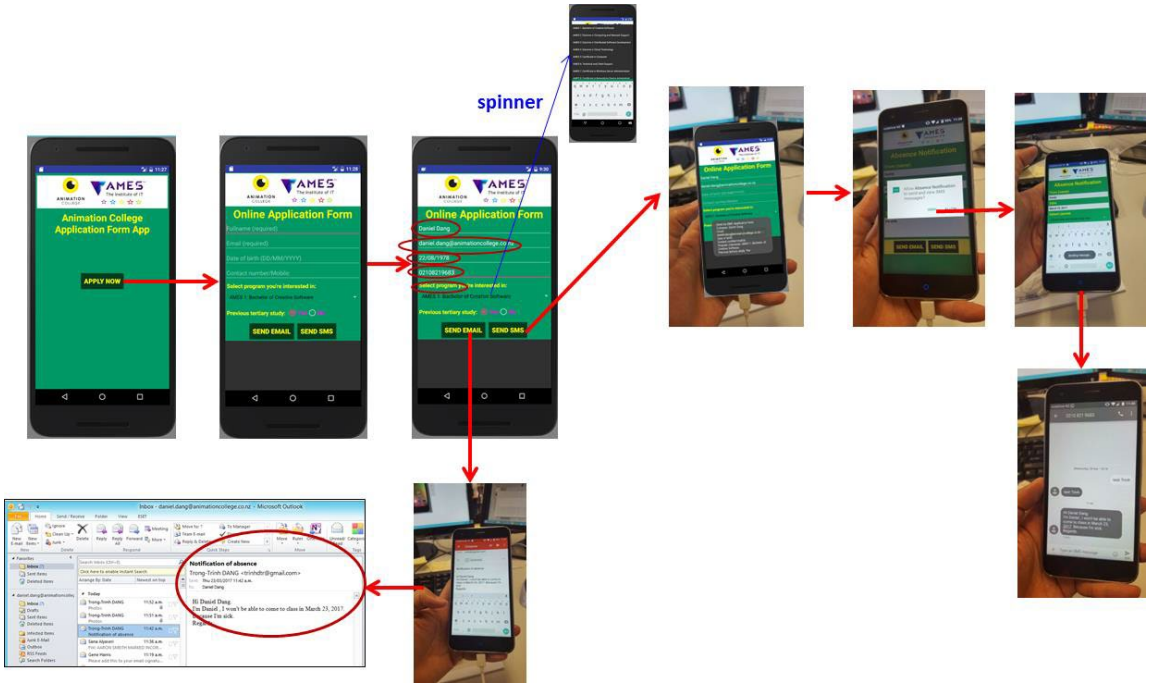
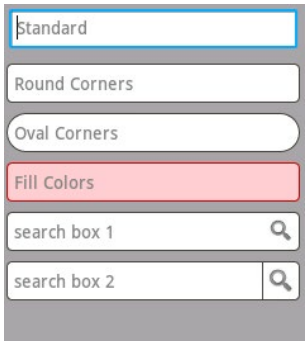


Figure 1: App wireframe

EditText Control

Source: https://www.tutorialspoint.com/android/android_edittext_control.htm

A EditText is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities.



STYLES OF EDIT TEXT

EditText Attributes

Following are the important attributes related to EditText control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class:

Sr.No	Attribute & Description
1	**android:autoText:** If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
2	**android:drawableBottom:** This is the drawable to be drawn below the text.
3	**android:drawableRight:** This is the drawable to be drawn to the right of the text.
4	**android:editable:** If set, specifies that this TextView has an input method.
5	**android:text:** This is the Text to display.

Inherited from **android.view.View** Class:

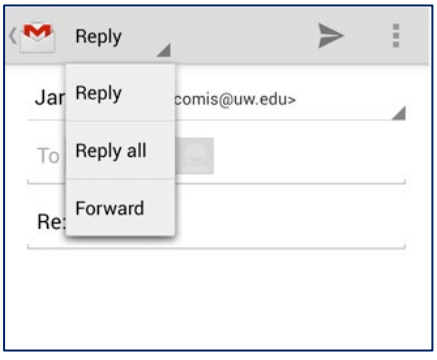
Sr.No	Attribute & Description
1	**android:background:** This is a drawable to use as the background.
2	**android:contentDescription:** This defines text that briefly describes content of the view.
3	**android:id:** This supplies an identifier name for this view.
4	**android:onClick:** This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility:** This controls the initial visibility of the view.

Spinner in Android

Source:
https://www.tutorialspoint.com/android/android_spinner_control.htm

Spinner allows you to select an item from a drop down menu

For example, when you are using Gmail application you would get drop down menu as shown below, you need to select an item from a drop down menu.



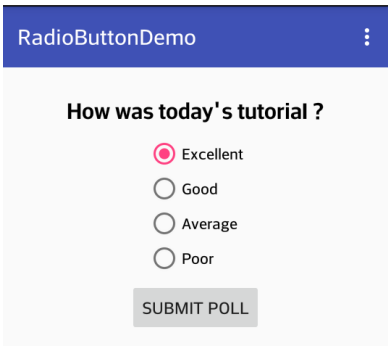
SPINNER EXAMPLE

Android - RadioGroup Control

Source:
https://www.tutorialspoint.com/android/android_radiogroup_control.htm

A RadioGroup class is used for set of radio buttons.

If we check one radio button that belongs to a radio group, it automatically unchecks any previously checked radio button within the same group.



RadioGroup Attributes

Following are the important attributes related to RadioGroup control. You can check **Android official documentation** for complete list of attributes and related methods which you can use to change these attributes are run time:

Attribute	Description
android:checkedButton	This is the id of child radio button that should be checked by default within this radio group.
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

Android - Sending SMS

Source: https://www.tutorialspoint.com/android/android_sending_sms.htm

In Android, you can use **SmsManager API** or **devices Built-in SMS application** to send SMS's.

SmsManager API:

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

Built-in SMS application:

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

Both need **SEND_SMS permission**.

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

Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below –

Sr.No.	Method & Description
1	**ArrayList<String> divideMessage(String text)** This method divides a message text into several fragments, none bigger than the maximum SMS message size.
2	**static SmsManager getDefault()** This method is used to get the default instance of the SmsManager

3	**void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent)** This method is used to send a data based SMS to a specific application port.
4	**void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents)** Send a multi-part text based SMS.
5	**void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent)** Send a text based SMS.

Android - Sending Email

Source: https://www.tutorialspoint.com/android/android_sending_email.htm

Email *is messages distributed by electronic means from one system user to one or more recipients via a network.*

Before starting Email Activity, You must know **Email functionality with intent**, Intent is carrying data from one component to another component with-in the application or outside the application.

To send an email from your application, you don't have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc. For this purpose, we need to write an Activity that launches an email client, using an implicit Intent with the right action and data. In this example, we are going to send an email from our app by using an Intent object that launches existing email clients.

Following section explains different parts of our Intent object required to send an email.

Intent Object - Action to send Email

You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

Intent Object - Data/Type to send Email

To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows:

```
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");
```

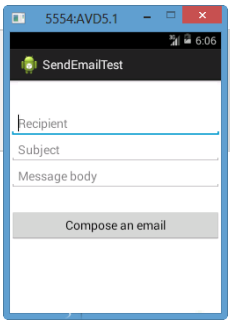
Intent Object - Extra to send Email

Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client. You can use following extra fields in your email –

Sr.No.	Extra Data & Description
1	**EXTRA_BCC:** A String[] holding e-mail addresses that should be blind carbon copied.
2	**EXTRA_CC:** A String[] holding e-mail addresses that should be carbon copied.
3	**EXTRA_EMAIL:** A String[] holding e-mail addresses that should be delivered to.
4	**EXTRA_HTML_TEXT:** A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.
5	**EXTRA_SUBJECT:** A constant string holding the desired subject line of a message.
6	**EXTRA_TEXT:** A constant CharSequence that is associated with the Intent, used with ACTION_SEND to supply the literal data to be sent.
7	**EXTRA_TITLE:** A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER.

Here is an example showing you how to assign extra data to your intent:

```
emailIntent.putExtra(Intent.EXTRA_EMAIL  , new String[]{"Recipient"});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject");
emailIntent.putExtra(Intent.EXTRA_TEXT   , "Message Body");
```



The output of above code is as shown in the beside image.

Read more:

[1]. Android Essentials: Creating Simple User Forms: https://code.tutsplus.com/tutorials/android-essentials-creating-simple-user-forms--mobile-1758

[2]. Activities: https://developer.android.com/guide/components/activities.html

[3]. Android Activity: https://www.tutorialspoint.com/android/android_acitivities.htm

[4]. Intents and Intent Filters: https://developer.android.com/guide/components/intents-filters.html

[5]. Intents and Filters: https://www.tutorialspoint.com/android/android_intents_filters.htm

[6]. Android – Sending SMS: https://www.tutorialspoint.com/android/android_sending_sms.htm

[7]. SmsManager: https://developer.android.com/reference/android/telephony/SmsManager.html

LAB 5: Android services (Music Player) & Listview (AC Program Manager App)

Services in Android
Lab 05: Play Music using MediaPlayer API

Objectives:

This exercise shows students to use the **Service component in Android**. Students will create their own Service Class (called myService.java) which opens and plays an audio file by using **MediaPlayer API**.

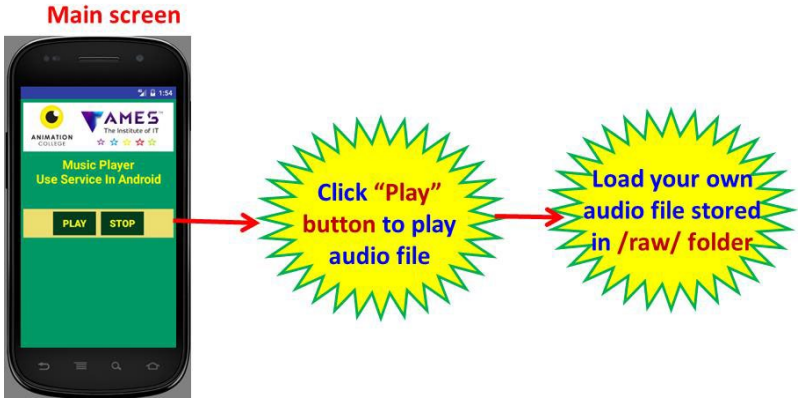


Figure 1: App wireframe

Services in Android:

Source: https://www.tutorialspoint.com/android/android_services.htm

In Android, A **service** is a component that runs in the background to perform long-running operations without needing to interact with the user and it works even if application is destroyed. A service can essentially take two states:

State	Description
Started	A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
Bound	A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

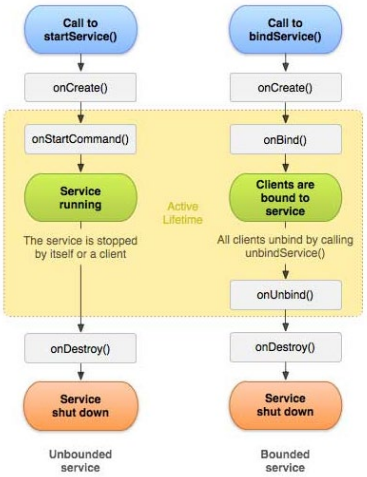
A Service in android is a background service which keeps running even after the parent application is closed. Popular use **cases:**

- For playing songs in background in a music player app
- For downloading a file in background from a server
- For showing the status of Connection to the chat server for a chat messenger app

A service has life cycle **callback methods** that you can implement to monitor changes in the service's state and you can perform work at the appropriate stage. The following diagram on the left shows the life cycle when the service is created with startService() and the diagram on the right shows the life cycle when the service is created with bindService():

To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. The **Service** base class defines various callback methods and the most important are given below. You don't need to implement all the callbacks methods.

However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.



Callback	Description
onStartCommand()	The system calls this method when another component, such as an activity, requests that the service be started, by calling *startService()*. If you implement this method, it is your responsibility to stop the service when its work is done, by calling *stopSelf()* or *stopService()*
onCreate()	The system calls this method when the service is first created using *onStartCommand()* or *onBind()*. This call is required to perform one-time set-up.
onDestroy()	The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

Android – MediaPlayer API

Source: https://www.tutorialspoint.com/android/android_mediaplayer.htm

Android provides many ways to **control playback of audio/video files and streams**. One of this way is through a class called **MediaPlayer**.

Android is providing MediaPlayer class to access built-in mediaplayer services like playing audio, video e.t.c. In order to use MediaPlayer, we have to **call a static Method create() of this class**. This method returns an instance of MediaPlayer class. Its syntax is as follows:

```
MediaPlayer mediaPlayer = MediaPlayer.create(this, R.raw.song);
```

The second parameter is the **name of the song that you want to play**. You have to make a new folder under your project with name **raw** and place the music file into it. Once you have created the Mediaplayer object you can call some methods to start or stop the music. These methods are listed below:

```
mediaPlayer.start();
mediaPlayer.pause();
```

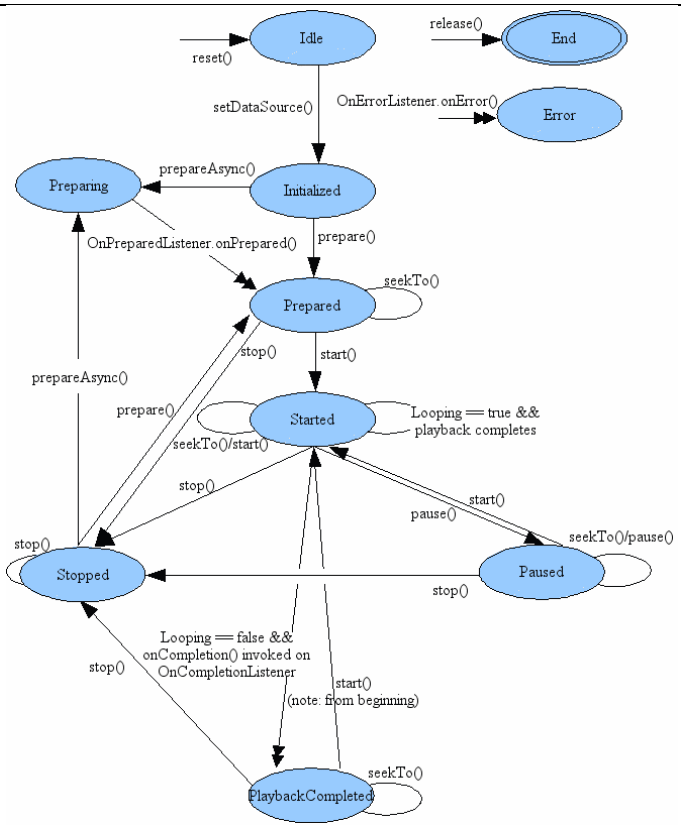
On call to **start()** method, the music will start playing from the beginning. If this method is called again after the **pause()** method, the music would start playing from where it is left and not from the beginning.

In order to start music from the beginning, you have to call **reset()** method. Its syntax is given below.

```
mediaPlayer.reset();
```

Apart from the start and pause method, there are other methods provided by this class for better dealing with audio/video files. These methods are listed below:

Sr.No	Method & description
1	**isPlaying():** This method just returns true/false indicating the song is playing or not
2	**seekTo(position):** This method takes an integer, and move song to that particular second
3	**getCurrentDuration():** This method returns the current position of song in milliseconds
4	**getDuration():** This method returns the total time duration of song in milliseconds
5	**reset():** This method resets the media player
6	**release():** This method releases any resource attached with MediaPlayer object
7	**setVolume(float leftVolume, float rightVolume):** This method sets the up down volume for this player
8	**setDataSource(FileDescriptor fd):** This method sets the data source of audio/video file
9	**selectTrack(int index):** This method takes an integer, and select the track from the list on that particular index
10	**getTrackInfo():** This method returns an array of track information



State diagram shows the life cycle and the states of a MediaPlayer object driven by the supported playback control operations.

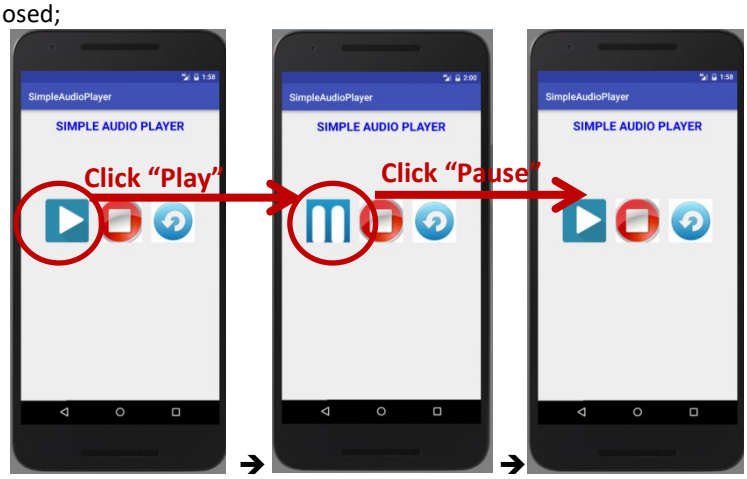
Read more:

[1]. Android – Service: https://www.tutorialspoint.com/android/android_services.htm
[2]. Sercices: https://developer.android.com/guide/components/services.html
[3]. Android – MediaPlayer: https://www.tutorialspoint.com/android/android_mediaplayer.htm
[4]. MediaPlayer API: https://developer.android.com/reference/android/media/MediaPlayer.html

Simple Music Player App:

In this exercise, you learn **audio related functions** of Android. You are able to construct a "**Simple audio player"** application using the **APIs** offered by Android (**MediaPlayer class**).

- This "Player" application plays an audio file stored **inside apk**;
- The Player application plays music by triggering **background service**. It means that player keeps playing music even if the main activity is closed;



The main functions of player application:

- **Play**: when clicking "play" button, we have to check whether a player instance (MediaPlayer object) is available (already created) or not available (not created yet)?
  1. If the player instance is not created yet (*play_reset = true*), create a media player instance using "**new**", and the player instance is put in "**Idle**" state.
     Call **setAudiostream()** method to set type of audi streaming by using AudioManager class;
     Call the **setDataSource()** method to pass the audio file path stored in sdcard and put the player instance into the "**Initialized**" state;
     Call **prepare()** method to move to "**Prepared**" state;
     Call **start()** method to play music and move to "**Started**" state;
     Finally, call **playPause()** method if it is not the first time clicking "Play" button. This playPause() method implements the "**playback**" (**resume playing**) when being in *"on Pausing" state* and "**pause**" when being in *"on Playing state".*
     Remember that this "Play" button image has two different states: "on Playing" and "on Pausing":
     - ✓ **"on Playing" state** shows that music is on playing. In order to support the "pause" function, it means If users click on "Pause" image, music streaming will be paused, the image displayed on "play" ImageButton will be "Pause" image (prompting that clicing this pause button to paus music streaming);
     - ✓ **"on Pausing" state** shows that music streaming is paused. In order to support the "playback" (resumg playing) function, it means that if users click on "Play" image, music stream will resume playing, the image displayed on "play" ImageButton will be "Play" image (prompting that "*Click this Play button to continue playing music streaming*");
  2. If the player instance is already created (*play_reset = false*), we call **playPause()** method;

- **Stop**: when clicking this "stop" button, we stop the completely the audio streaming. We have to check if the player instance is available (created) or not.
  1. If the player instance is already created (*play_reset = false*), call **stop() method** and change the image of "play" button to "play image" and move to "**Stopped**" state. We will call **Prepare() method** to move to "**Prepared**" state. The player instance will stand by at the "beginning" of audio streaming (re-play the song/music).
  2. If the player instance is not created (*play_reset = true*), we do nothing.

- **Reset**: when clicking "reset" button, the player (MediaPlayer object) is released so that resources used by the internal player engine associated with the MediaPlayer object (player) can be released immediately. Resource may include singleton resources such as hardware acceleration components. Faillure to call **release()** may cause subsequenct instances of MediaPlayer objects to fallback to software implementations or fail altogether. Once the MediaPlayer object (player) is in the "**End**" state, it can no longer be used and there is no way to bring it back to any other state. We have to check if the player instance is available (created) or not.
  1. If the player instance is already created (*play_reset = false*), call **reset() method** and change the image of "play" button to "play image" and move to "**Idle**" state. The player instance will stand by at "**Idle**"state. The "play_reset" variable now is set to "true" indicating "no player instance available".
  2. If the player instance is not created (*play_reset = true*), we do nothing.

***Note**: We use the "**play_reset**" variable to indicate whether a media player is already created (false) or not yet created (true).*

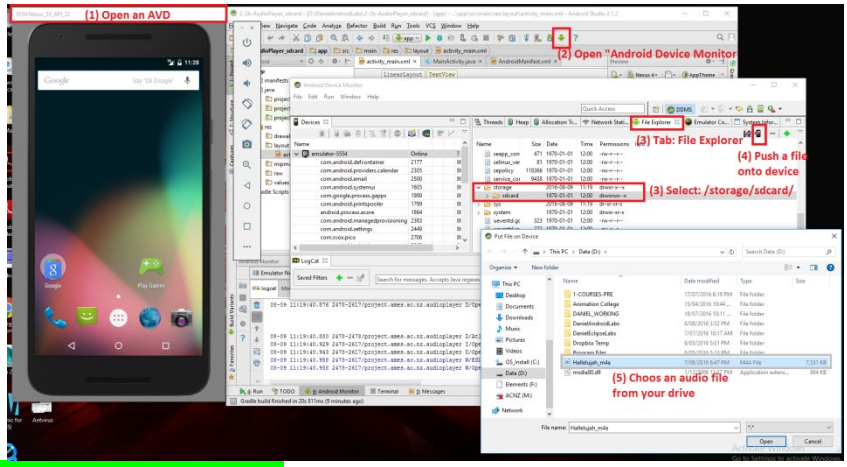
Playback control of audio/video files and streams is managed as a state machine.
The following diagram shows the life cycle and the states of a MediaPlayer object driven by the supported **playback** control operations.

- ✓ The ovals represent the states a MediaPlayer object may reside in.
- ✓ The arcs represent the playback control operations that drive the object state transition. There are two types of arcs. The arcs with a single arrow head represent synchronous method calls, while those with a double arrow head represent asynchronous method calls.

Push an audio file into Sdcard

Prepare audio file: Import (audio) files into the Sdcard on emulator in android

1. Open **an Emulator** (AVD);
2. Open **Android Device Monitor (ADM)**;
3. Make sure the "**File Explorer**" is selected;
4. Scroll down and select **/storage/sdcard/;**
5. Click on "***Push file into sdcard***" icon at top right;
6. Choose the file (**vnmusic_tuyhungquacau.mp3**) to import to **/storage/sdcard/;**



Modify the MainActivity.java as following

+ In order to open an audio file stored in Sdcard, we will modify the onClick() method, the "*case R.id.play*:" as following:

- If no player instance is available, then create a media player first;
- Initialize the "player" instance and then play an audio file stored in storage/sdcard/;
- Create a "player" instance - move **to "Idle" state**;
- Set type of audio stream;
- Declare the PATH to the audio file;
- Move "player" object to **"Initialized" state**;
- Move "player" object to "**Prepared" state**;
- Start to play audio file in streaming, move to **"Started" state**;
- Change to **"pause" image**;
- Set property (looping) of the "player" instance: false = no looping;

ListView and XML Parser in Android
Lab 05_2: ACProgramManagerApp

Objectives

In this lab, you will learn how to **implement a ListView**,how to **parse the XML file** and extr**act necessary information from it by using XMLPullParse class**, then display information on **screen (in Table format).**
You also learn the use of **XMLPullParser class** to create a **basic Weather application** that allows you to **parse XML from google weather API** and show the result.

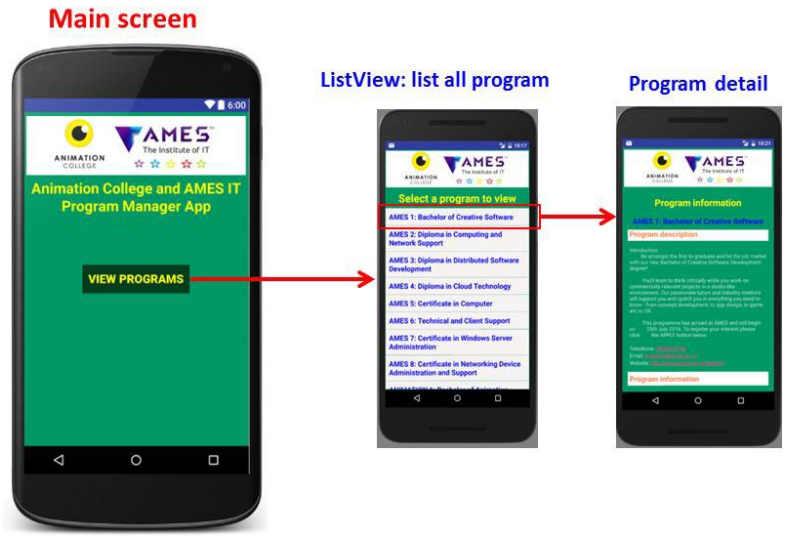
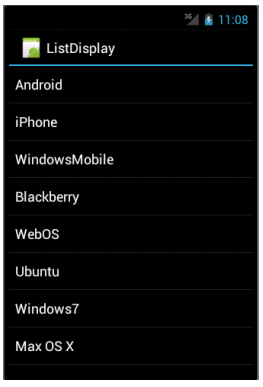


Figure 1: App wireframe

ListView

Source: https://www.tutorialspoint.com/android/android_list_view.htm

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.



LIST VIEW

ListView Attributes

Following are the important attributes specific to GridView −

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:divider	This is drawable or color to draw between list items. .
android:dividerHeight	This specifies height of the divider. This could be in px, dp, sp, in, or mm.
android:entries	Specifies the reference to an array resource that will populate the ListView.
android:footerDividersEnabled	When set to false, the ListView will not draw the divider before each footer view. The default value is true.
android:headerDividersEnabled	When set to false, the ListView will not draw the divider after each header view. The default value is true.

An adapter actually bridges between UI components and the data source that fill data into UI Component. The Adapter holds the data and sends the data to adapter view, the view can take the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.
The **ListView** is subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

ArrayAdapter
You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array:

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

Here are arguments for this constructor:

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows:

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file.

Read more:
[1]. List View: https://developer.android.com/guide/topics/ui/layout/listview.html
[2]. Android List View: https://www.tutorialspoint.com/android/android_list_view.htm

XML parser

Source: **XML Parser Tutorial**: http://www.tutorialspoint.com/android/android_xml_parsers.htm

XML stands for Extensible Mark-up Language.XML is a very popular format and commonly used for sharing data on the internet. This chapter explains how to parse the XML file and extract necessary information from it.
Android provides three types of XML parsers which are **DOM, SAX and XMLPullParser**. Among all of them android recommend XMLPullParser because it is efficient and easy to use. So we are going to use XMLPullParser for parsing XML.
The first step is to identify the fields in the XML data in which you are interested in. For example. In the XML given below we interested in getting temperature only:

```
<?xml version="1.0"?>
<current>

  <city id="2643743" name="London">
    <coord lon="-0.12574" lat="51.50853"/>
    <country>GB</country>
    <sun rise="2013-10-08T06:13:56" set="2013-10-08T17:21:45"/>
  </city>

  <temperature value="289.54" min="289.15" max="290.15" unit="kelvin"/>
  <humidity value="77" unit="%"/>
  <pressure value="1025" unit="hPa"/>
</current>
```

XML - Elements
An xml file consist of many components. Here is the table defining the components of an XML file and their description:

Sr.No	Component & description
1	**Prolog:** An XML file starts with a prolog. The first line that contains the information about a file is prolog
2	**Events:** An XML file has many events. Event could be like this. Document starts , Document ends, Tag start , Tag end and Text e.t.c
3	**Text:** Apart from tags and events, and xml file also contains simple text. Such as **GB** is a text in the country tag.
4	**Attributes:** Attributes are the additional properties of a tag such as value e.t.c

XML - Parsing

In the next step, we will create XMLPullParser object, but in order to create that we will first create XmlPullParserFactory object and then call its newPullParser() method to create XMLPullParser. Its syntax is given below –

```
private XmlPullParserFactory xmlFactoryObject = XmlPullParserFactory.newInstance();
private XmlPullParser myparser = xmlFactoryObject.newPullParser();
```

The next step involves specifying the file for XmlPullParser that contains XML. It could be a file or could be a Stream. In our case it is a stream.Its syntax is given below –

```
myparser.setInput(stream, null);
```

The last step is to parse the XML. An XML file consist of events, Name, Text, AttributesValue e.t.c. So XMLPullParser has a separate function for parsing each of the component of XML file. Its syntax is given below:

```
int event = myParser.getEventType();
while (event != XmlPullParser.END_DOCUMENT)
{
  String name=myParser.getName();
  switch (event){
    case XmlPullParser.START_TAG:
    break;

    case XmlPullParser.END_TAG:
    if(name.equals("temperature")){
      temperature = myParser.getAttributeValue(null,"value");
    }
    break;
  }
  event = myParser.next();
}
```

The method **getEventType** returns the type of event that happens. e.g: Document start , tag start e.t.c. The method **getName** returns the name of the tag and since we are only interested in temperature , so we just check in conditional statement that if we got a temperature tag , we call the method**getAttributeValue** to return us the value of temperature tag.

Apart from these methods, there are other methods provided by this class for better parsing XML files. These methods are listed below:

Sr.No	Method & description
1	**getAttributeCount():** This method just Returns the number of attributes of the current start tag

2	**getAttributeName(int index):** This method returns the name of the attribute specified by the index value
3	**getColumnNumber():** This method returns the Returns the current column number, starting from 0.
4	**getDepth():** This method returns Returns the current depth of the element.
5	**getLineNumber():** Returns the current line number, starting from 1.
6	**getNamespace():** This method returns the name space URI of the current element.
7	**getPrefix():** This method returns the prefix of the current element
8	**getName():** This method returns the name of the tag
9	**getText():** This method returns the text for that particular element
10	**isWhitespace():** This method checks whether the current TEXT event contains only whitespace characters.

REFERENCES:

[1]. Android - XML Parser Tutorial: http://www.tutorialspoint.com/android/android_xml_parsers.htm

[2]. Android - XML Parser Tutorial use of XMLPullParser parse xml on web api|android studio: https://www.youtube.com/watch?v=FQKmFCAYcmA&index=20&list=PLYDICjs3cYp_uBzvBmHZN_H6clYs7hq7W

[3]. Create a Weather App on Android: http://code.tutsplus.com/tutorials/create-a-weather-app-on-android--cms-21587

Extra work

This exercise is an example demonstrating the use of XMLPullParser class. It creates a basic Weather application that allows you to parse XML from google weather API and show the result.

RSS (Really Simple Syndication) FEED

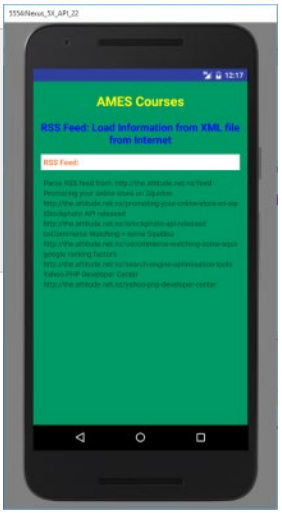
Source: Android - RSS Reader Tutorial: https://www.tutorialspoint.com/android/android_rss_reader.htm

RSS stands for Really Simple Syndication. RSS is an easy way to share your website updates and content with your users so that users might not have to visit your site daily for any kind of updates.

Example of RSS FEED: http://the.attitude.net.nz/feed

RSS is a document that is created by the website with .xml extension. You can easily parse this document and show it to the user in your application. An RSS document looks like this.

```
<rss version="2.0">
<channel>
   <title>Sample RSS</title>
   <link>http://www.google.com</link>
   <description>World's best search engine</description>
</channel>
</rss>
```



RSS Elements

An RSS document such as above has the following elements.

Sr.No	Component & description
1	**Channel:** This element is used to describe the RSS feed
2	**Title:** Defines the title of the channel
3	**Link:** Defines the hyper link to the channel
4	**Description:** Describes the channel

Parsing RSS

Parsing an RSS document is more like parsing XML. So now lets see how to parse an XML document.
For this, We will create XMLPullParser object , but in order to create that we will first create XmlPullParserFactory object and then call its newPullParser() method to create XMLPullParser. Its syntax is given below –

```
private XmlPullParserFactory xmlFactoryObject = XmlPullParserFactory.newInstance();
private XmlPullParser myparser = xmlFactoryObject.newPullParser();
```

The next step involves specifying the file for XmlPullParser that contains XML. It could be a file or could be a Stream. In our case it is a stream.Its syntax is given below –

```
myparser.setInput(stream, null);
```

The last step is to parse the XML. An XML file consist of events , Name , Text , AttributesValue e.t.c. So XMLPullParser has a separate function for parsing each of the component of XML file. Its syntax is given below –

```
int event = myParser.getEventType();
while (event != XmlPullParser.END_DOCUMENT)
{
   String name=myParser.getName();

   switch (event){
      case XmlPullParser.START_TAG:
      break;

      case XmlPullParser.END_TAG:
      if(name.equals("temperature")){
         temperature = myParser.getAttributeValue(null,"value");
```

```
            }
            break;
        }
        event = myParser.next();
    }
```

The method **getEventType** returns the type of event that happens. e.g: Document start , tag start e.t.c. The method **getName** returns the name of the tag and since we are only interested in temperature , so we just check in conditional statement that if we got a temperature tag , we call the method**getAttributeValue** to return us the value of temperature tag.

Apart from these methods, there are other methods provided by this class for better parsing XML files. These methods are listed below:

Sr.No	Method & description
1	**getAttributeCount():** This method just Returns the number of attributes of the current start tag.
2	**getAttributeName(int index):** This method returns the name of the attribute specified by the index value.
3	**getColumnNumber():** This method returns the Returns the current column number, starting from 0.
4	**getDepth():** This method returns Returns the current depth of the element.
5	**getLineNumber():** Returns the current line number, starting from 1.
6	**getNamespace():** This method returns the name space URI of the current element.
7	**getPrefix():** This method returns the prefix of the current element.
8	**getName():** This method returns the name of the tag.
9	**getText():** This method returns the text for that particular element.
10	**isWhitespace():** This method checks whether the current TEXT event contains only white space characters.

Android – Creating links using linkify

Source: http://www.aviyehuda.com/blog/2011/01/27/android-creating-links-using-linkfy/

Linkify is a class that lets you create links from a TextView or a Spannable.
*You can create links not just to **web pages**, but also to **locations on the map**, **emails** and even **phone numbers**.*

LAB 6: Broadcast and Content Providers

BroadCast Receivers component in Android
Lab 06_1: BroadcastMessage App

Objectives

Students will learn the Android Broadcast Receivers component to send a broadcast message.

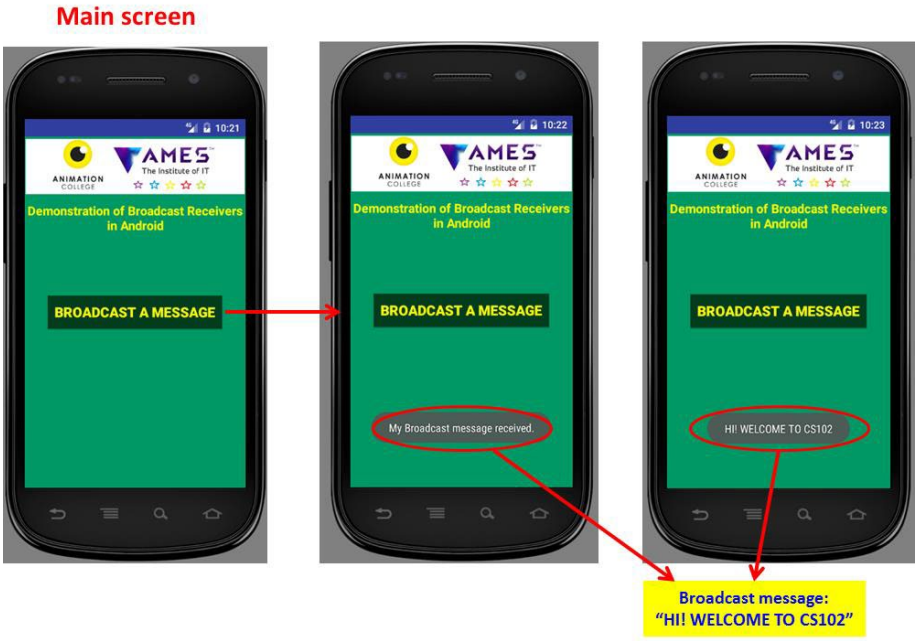


Figure 1: Wireframe of broadcast message app

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
There are following two important steps to make BroadcastReceiver works for the system broadcasted intents –

- Creating the Broadcast Receiver.
- Registering Broadcast Receiver

There is one additional steps in case you are going to implement your custom intents then you will have to create and broadcast those intents.

Android BroadcastReceivers

Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

```
public class MyReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
  }
}
```

There are several system generated events defined as final static fields in the **Intent** class. The following table lists a few important system events.

Sr.No	Event Constant & Description
1	**android.intent.action.BATTERY_CHANGED** Sticky broadcast containing the charging state, level, and other information about the battery.
2	**android.intent.action.BATTERY_LOW:** Indicates low battery condition on the device.
3	**android.intent.action.BATTERY_OKAY:** Indicates the battery is now okay after being low.
4	**android.intent.action.BOOT_COMPLETED:** This is broadcast once, after the system has finished booting.
5	**android.intent.action.BUG_REPORT:** Show activity for reporting a bug.
6	**android.intent.action.CALL:** Perform a call to someone specified by the data.
7	**android.intent.action.CALL_BUTTON:** The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
8	**android.intent.action.DATE_CHANGED:** The date has changed.
9	**android.intent.action.REBOOT:** Have the device reboot.

Broadcasting Custom Intents

If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the *sendBroadcast()* method inside your activity class. If you use the *sendStickyBroadcast(Intent)* method, the Intent is **sticky**, meaning the *Intent* you are sending stays around after the broadcast is complete.

```
//Create the BroadcastReceiver containing the broadcast message
Intent broadcast_message = new Intent();
broadcast_message.putExtra("MyBroadcastMessage", "HI! WELCOME TO CS102");
broadcast_message.setAction("nz.ac.bcs.cs102.MyBroadcastMessage");
//Send  broadcast message
sendBroadcast(broadcast_message);
```

This intent *nz.ac.bcs.cs102.MyBroadcastMessage* can also be registered in similar way as we have regsitered system generated intent.

```
<!--Add MyBroadcastReceiver-->
<receiver android:name=".MyBroadcastReceiver">
    <!--First intent filter to receive nz.ac.bcs.cs102.MyBroadcastMessage-->
    <intent-filter android:priority="1000000">
        <action android:name="nz.ac.bcs.cs102.MyBroadcastMessage"></action>
    </intent-filter>
    <!--Second intent filter to receive nz.ac.bcs.cs102.FileDownloadFinished-->
    <intent-filter android:priority="1000000">
        <action android:name="nz.ac.bcs.cs102.FileDownloadFinished"></action>
    </intent-filter>
</receiver>
```

Read more:

[1]. Android - Broadcast Receivers: https://www.tutorialspoint.com/android/android_useful_resources.htm
[2]. Android BroadcastReceiver - Tutorial: http://www.vogella.com/tutorials/AndroidBroadcastReceiver/article.html
[3]. Incomming sms broadcast receiver: http://androidexample.com/Incomming_SMS_Broadcast_Receiver_-_Android_Example/index.php?view=article_discription&aid=62#
[4]. Introduction to broadcast receiver basics: http://androidexample.com/Introduction_To_Broadcast_Receiver_Basics/index.php?view=article_discription&aid=60#
[5]. Understanding Android Broadcast Receivers: http://codetheory.in/android-broadcast-receivers/
[6]. Android Broadcast Intents and Broadcast Receivers: http://www.techotopia.com/index.php/Android_Broadcast_Intents_and_Broadcast_Receivers

Content Providers in Android: Shared Preferences
Lab 06_2 NoteTaker App

Objectives:

In this lab, you will develop a "TakeNote" app using **Content Provider component** in Android: **Shared Preferences.** This example demonstrates the use of the Shared Preferences to develop an app to write notes and save in a file. The notes are saved when the application is closed and brought back when it is opened again.

- Save anything that crosses your mind and store all useful information with the NoteTaker app.
- To help you organizing yourself, suitable for adding personal notes, endless shopping lists, lists of things that you need to take a trip, to-do lists, etc.

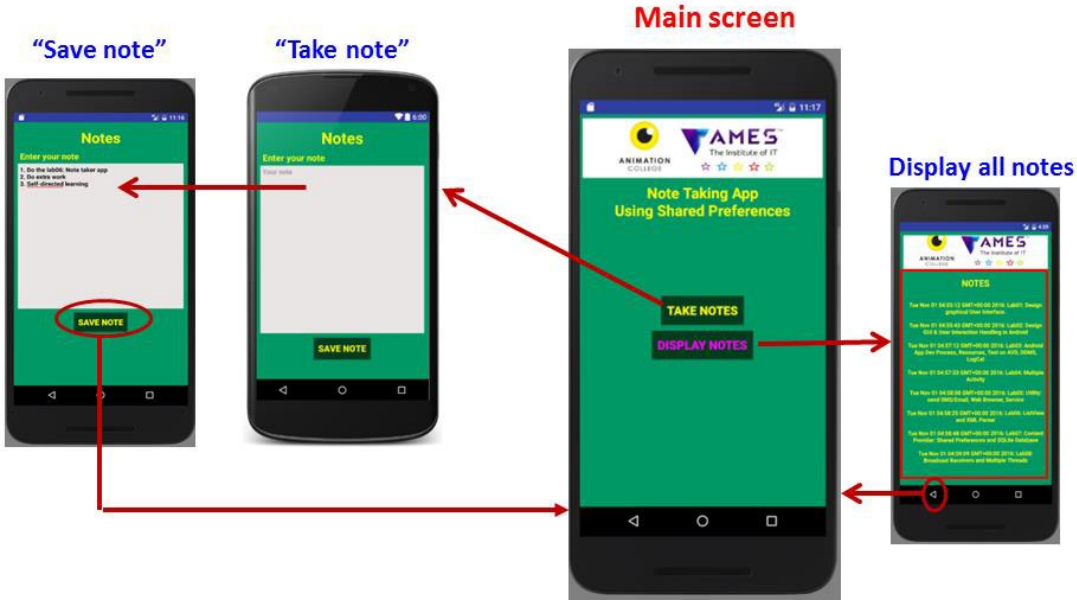
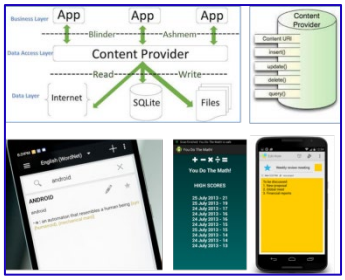


Figure 1: App wireframe

Shared Preferences in Android:

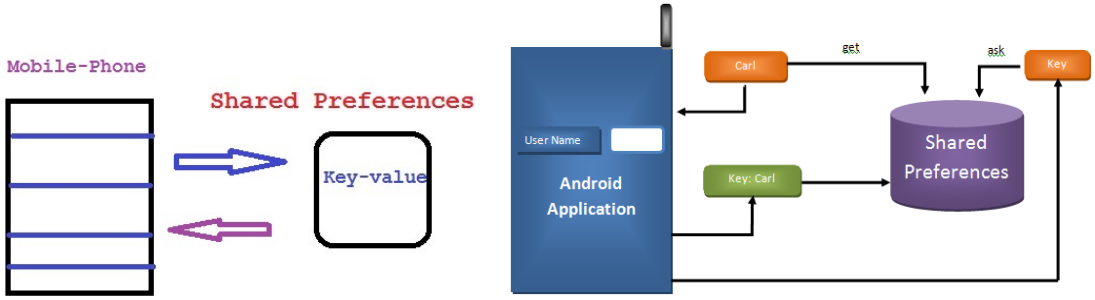
A **Content Provider component** supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

- Content providers let you centralize content in one place and have many different applications access it as needed.
- A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases, this data is stored in SQlite database.



Some examples of Cotent Provider Applications

Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of [key,value] pair.



In order to use shared preferences, you have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

```
SharedPreferences sharedpreferences = getSharedPreferences(MyPREFERENCES,
Context.MODE_PRIVATE);
```

The first parameter is the key and the second parameter is the MODE. Apart from private there are other modes available that are listed below:

Sr.No	Mode and description
1	**MODE_APPEND:** This will append the new preferences with the already existing preferences
2	**MODE_ENABLE_WRITE_AHEAD_LOGGING:** Database open flag. When it is set , it would enable write ahead logging by default
3	**MODE_MULTI_PROCESS:** This method will check for modification of preferences even if the sharedpreference instance has already been loaded
4	**MODE_PRIVATE:** By setting this mode , the file can only be accessed using calling application
5	**MODE_WORLD_READABLE:** This mode allow other application to read the preferences
6	**MODE_WORLD_WRITEABLE:** This mode allow other application to write the preferences

You can save something in the sharedpreferences by using SharedPreferences.Editor class. You will call the edit method of SharedPreference instance and will receive it in an editor object. Its syntax is:

```
Editor editor = sharedpreferences.edit();
editor.putString("key", "value");
editor.commit();
```

Apart from the putString method, there are methods available in the editor class that allows manipulation of data inside shared preferences. They are listed as follows:

Sr. no	Mode and description
1	**apply():** It is an abstract method. It will commit your changes back from editor to the

	sharedPreference object you are calling
2	**clear():** It will remove all values from the editor
3	**remove(String key):** It will remove the value whose key has been passed as a parameter
4	**putLong(String key, long value):** It will save a long value in a preference editor
5	**putInt(String key, int value):** It will save a integer value in a preference editor
6	**putFloat(String key, float value):** It will save a float value in a preference editor

References:

+ Android - Content Providers: https://www.tutorialspoint.com/android/android_content_providers.htm
+ Android - Shared Preferences Tutorial: https://www.tutorialspoint.com/android/android_shared_preferences.htm

Improve the lab02_AndroidQuizApp:

- In the main screen, there are two options (buttons): "Quiz" and "Top 5 scores".
- When clicking "Quiz", the app opens a new activity (new screen) displaying the quiz question and 4 possible answers, along with the "Score";
- When clicking "Top 3 best scores", the app opens a new activity (new screen) displaying the scores. If possible, it displays only top 3 best scores.

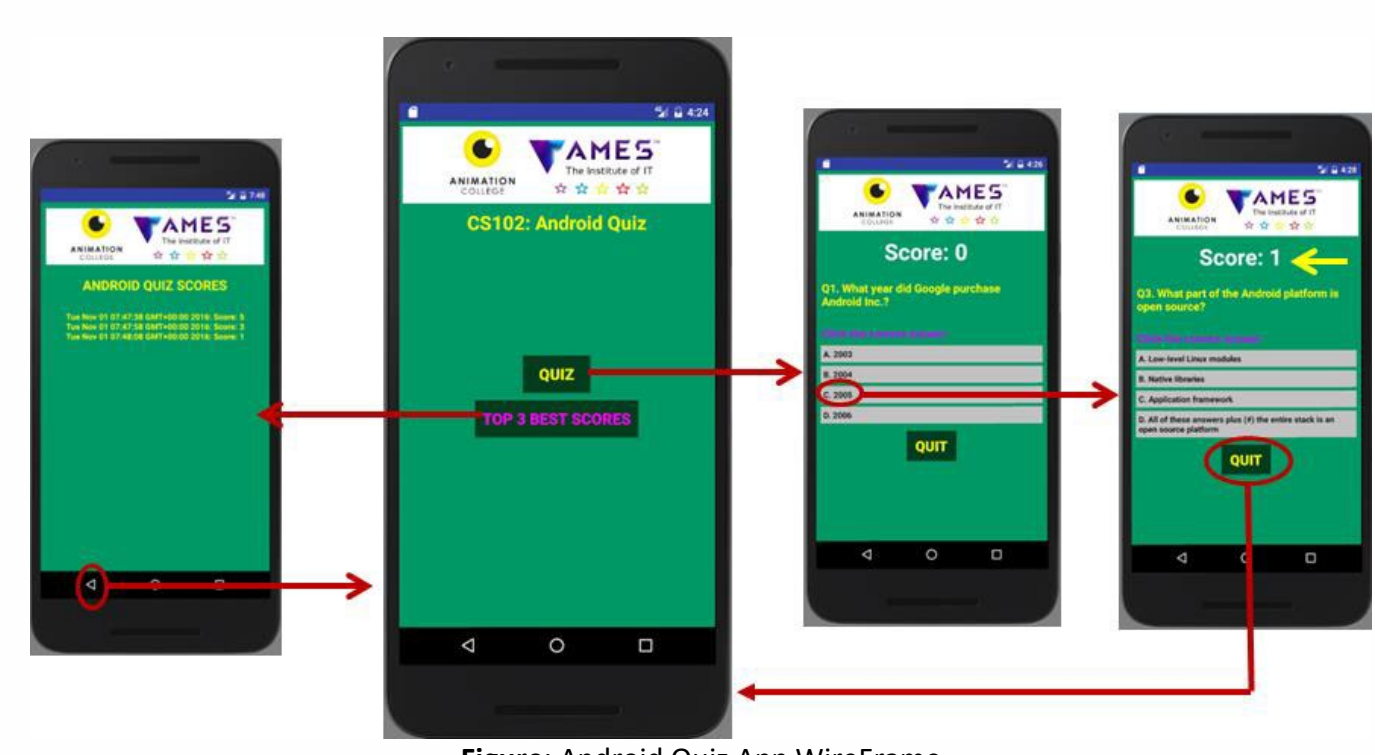


Figure: Android Quiz App WireFrame

LAB 7: Multiple Thread, Network and Internet

Multiple Thread in Android
Lab 07_1: ProgressBar and Huge Computation

Objectives

In this lab, we will see the problems encountered in doing long-rung processing on the main UI thread. We then examine two different approaches to create a multi-threaded application using threads/handlers and using AsynTask, both of which permit long-running processing to be done in the background, while at the same time permit the UI to be updated regularly on the progress.

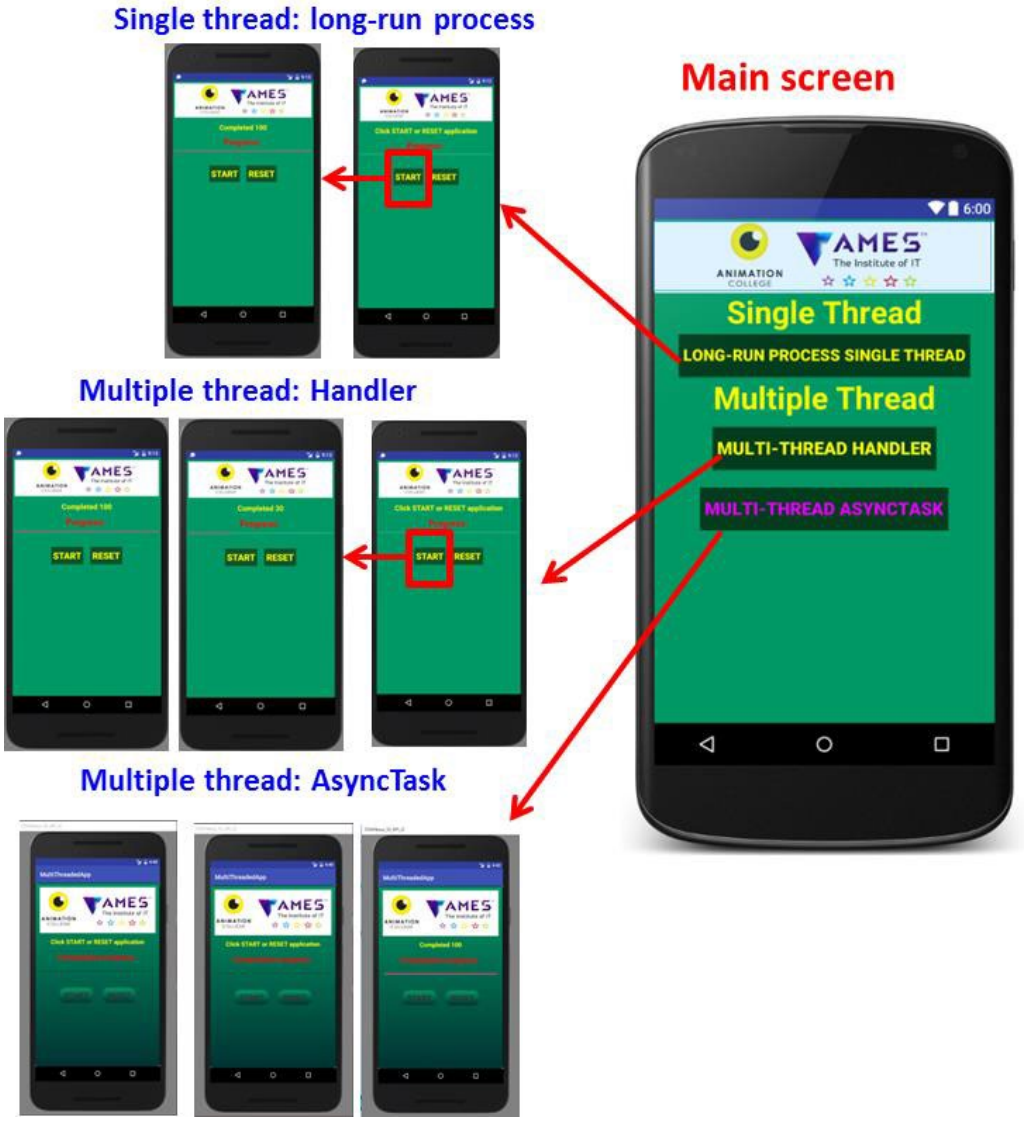
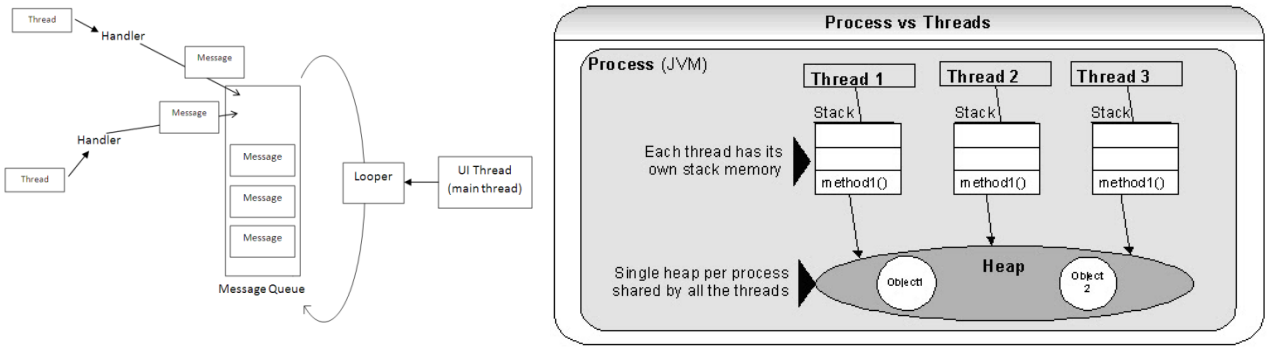


Figure 1: App wireframe

Multiple-thread



Android Multiple Threads

Multi-threading is defined as a feature through which we can run two or more concurrent threads of a process. In this a process, the common data is shared among all these threads also known as sub-processes exclusively. In android there are many ways through which multi-threading can be established in the application.

Multi-Threading in Android is a unique feature through which more than one threads execute together without hindering the execution of other threads. Multi-Threading in Android is not different from conventional multi-Threading. A class can be thought of as a process having its method as it's sub-processes or threads. All these methods can run concurrently by using feature of Multi-Threading. In android, multi-Threading can be achieved through the use of **many in-built classes**. Out of them, **Handler** class is most commonly used.

Handler class in Android:

- **Handler class** come from the **Package android.os.Handler package** and is most commonly used for multi-threading in android. Handler class provide sending and receiving feature for messages between different threads and handle the thread execution which is associated with that instance of Handler class.
- In android class, every thread is associated with an instance of Handler class and it allows the thread to run along with other threads and communicate with them through messages.

Runnable Interface in Android:

- **Runnable interface** is used in multi-threading to be called in a loop when the thread starts. It is a type of thread that executes the statement in its body or calls other methods for a specified or infinite number of times.
- This **runable interface** is used by **the Handler class** to execute the multi-threading, i.e., to execute one or more thread in specified time.
- **Runnable** is an interface which is implemented by the class desired to support multithreading and that class must implements it's abstract method public void run().
- **Run() method** is the core of multithreading as it includes the statement or calls to other methods that the thread needs to be made for multithreading.

AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a **helper class** around Thread and Handler and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the java.util.concurrent package such as Executor, ThreadPoolExecutor and FutureTask.

An **asynchronous task** is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate andonPostExecute.

Threads: AsyncTask

- Threads with Loopers and Handlers provides a powerful framework:
  - Code gets complicated and difficult to read;
  - For complex operations that require frequent UI updates;
- AsyncTask: simplifies the creation of long-running tasks that need to communicate with the UI:
  - Takes care of thread management;
  - Has to be created on the UI thread and executed only once;
- AsyncTask has the following excellent features:
  - Ability to **return values** of custom type to the UI thread when the task finished;
  - Ability to execute some code **in the UI thread** before the background task begins execution and after it finished;
  - Ability to **push updates** to the UI thread during the execution of the background task;
  - **Automatic** under-the-hood thread management;
- Several methods are parts of AsyncTask:
  - **doInBackground()** executes automatically on a worker thread;
  - **onPreExecute(), onPostExecute(),** and **onProgressUpdate()** are all invoked on the UI thread;
  - Call **publishProgress()** at any time in **doInBackground()** to execute **onProgressUpdate()** on the UI thread;
  - The **value** returned by **doInBackground()** is sent to **onPostExecute();**
  - You can cancel the task at any time, from any thread;
- NOTE: Don't confuse the word "**Task**" used here with "Task" as we used it in defining "**Back Stack/Activity Stack**" earlier;

When an asynchronous task is executed, the task goes through 4 steps:

1. onPreExecute()
2. doInBackground(Params...)
3. onProgressUpdate(Progress...)
4. onPostExecute(Result)

Read more:
[1]. Java - Multithreading: https://www.tutorialspoint.com/java/java_multithreading.htm
[2]. How to implement Multi-Threading in android with Handler Class: http://mrbool.com/how-to-implement-multi-threading-in-android-with-handler-class/28175#ixzz4GJOQ78g6

Network connection and Internet in Android
Lab 07_2: Network Connection and Download File

Objectives

In this exercise, students learn to use HttpURLConnection class to **download HTML files** (audio file, image file, and text file) from a given web page.

Android - Network Connection

Source: https://www.tutorialspoint.com/android/android_network_connection.htm

Android lets your application connect to the internet or any other local network and allows you to perform network operations.
A device can have various types of network connections. This chapter focuses on using either a Wi-Fi or a mobile network connection.

Checking Network Connection

Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c. For this android provides **ConnectivityManager** class. You need to instantiate an object of this class by calling **getSystemService()** method. Its syntax is given below:

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

Once you instantiate the object of ConnectivityManager class, you can use **getAllNetworkInfo** method to get the information of all the networks. This method returns an array of **NetworkInfo**. So you have to receive it like this.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

The last thing you need to do is to check **Connected State** of the network. Its syntax is given below:

```
for (int i = 0; i<info.length; i++){
  if (info[i].getState() == NetworkInfo.State.CONNECTED){
    Toast.makeText(context, "Internet is connected
    Toast.LENGTH_SHORT).show();
  }
}
```

Apart from this connected states, there are other states a network can achieve. They are listed below:

Sr.No	State

1	Connecting
2	Disconnected
3	Disconnecting
4	Suspended
5	Unknown

Performing Network Operations

After checking that you are connected to the internet, you can perform any network operation. Here we are fetching the html of a website from a url.

Android provides **HttpURLConnection** and **URL** class to handle these operations. You need to instantiate an object of URL class by providing the link of website. Its syntax is as follows:

```
String link = "http://www.google.com";
URL url = new URL(link);
```

After that you need to call **openConnection** method of url class and receive it in a HttpURLConnection object. After that you need to call the **connect** method of HttpURLConnection class.

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.connect();
```

And the last thing you need to do is to fetch the HTML from the website. For this you will use **InputStream** and **BufferedReader** class. Its syntax is given below:

```
InputStream is = conn.getInputStream();
BufferedReader reader = new BufferedReader(new InputStreamReader(is, "UTF-8"));
String webPage = "",data="";

while ((data = reader.readLine()) != null){
  webPage += data + "\n";
}
```

Apart from this connect method, there are other methods available in HttpURLConnection class. They are listed below:

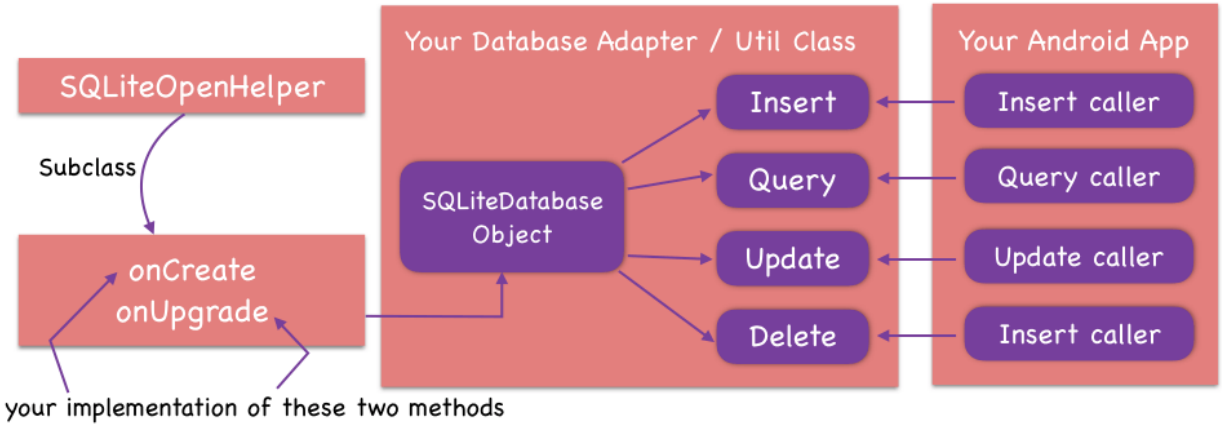
Sr.No	Method & description
1	**disconnect()** This method releases this connection so that its resources may be either reused or closed
2	**getRequestMethod()** This method returns the request method which will be used to make the request to the remote HTTP server
3	**getResponseCode()** This method returns response code returned by the remote HTTP server
4	**setRequestMethod(String method)** This method Sets the request command which will be sent to the remote HTTP server
5	**usingProxy()** This method returns whether this connection uses a proxy server or not

LAB 8: SQ Lite Database

Android Content Providers: SQ LITE Database
Lab 08: CS102LabJournal Manager

Objectives

In software applications, it is mostly required to save information for some internal use or off course to provide user to great features depending on the data. And when we talk about android so SQLite is that default feature which is used as a database and also used as a local database for any application. This tutorial shows a very simple example which is to just store important data like shops address or contacts using SQLite **Database** in the android studio.



Android provides many ways to store data, SQLite Database is one of them that is already include in android OS. We have to just simply use it according to our need.

Here, we will take a simple example to store CS102 lab journals (Android Mobile App).

"Lab journals" table structure:

Field	Data type	Key
Student ID	**String**	**Primary**
Fullname	String	
Email	String	
Date of birth	String	
Lab01	Integer: not submitted =0, submitted=1	
Lab02	Integer: not submitted =0, submitted=1	
Lab03	Integer: not submitted =0, submitted=1	
Lab04	Integer: not submitted =0, submitted=1	
Lab05	Integer: not submitted =0, submitted=1	
Lab06	Integer: not submitted =0, submitted=1	
Lab07	Integer: not submitted =0, submitted=1	
Lab08	Integer: not submitted =0, submitted=1	

Below is the app wireframe:

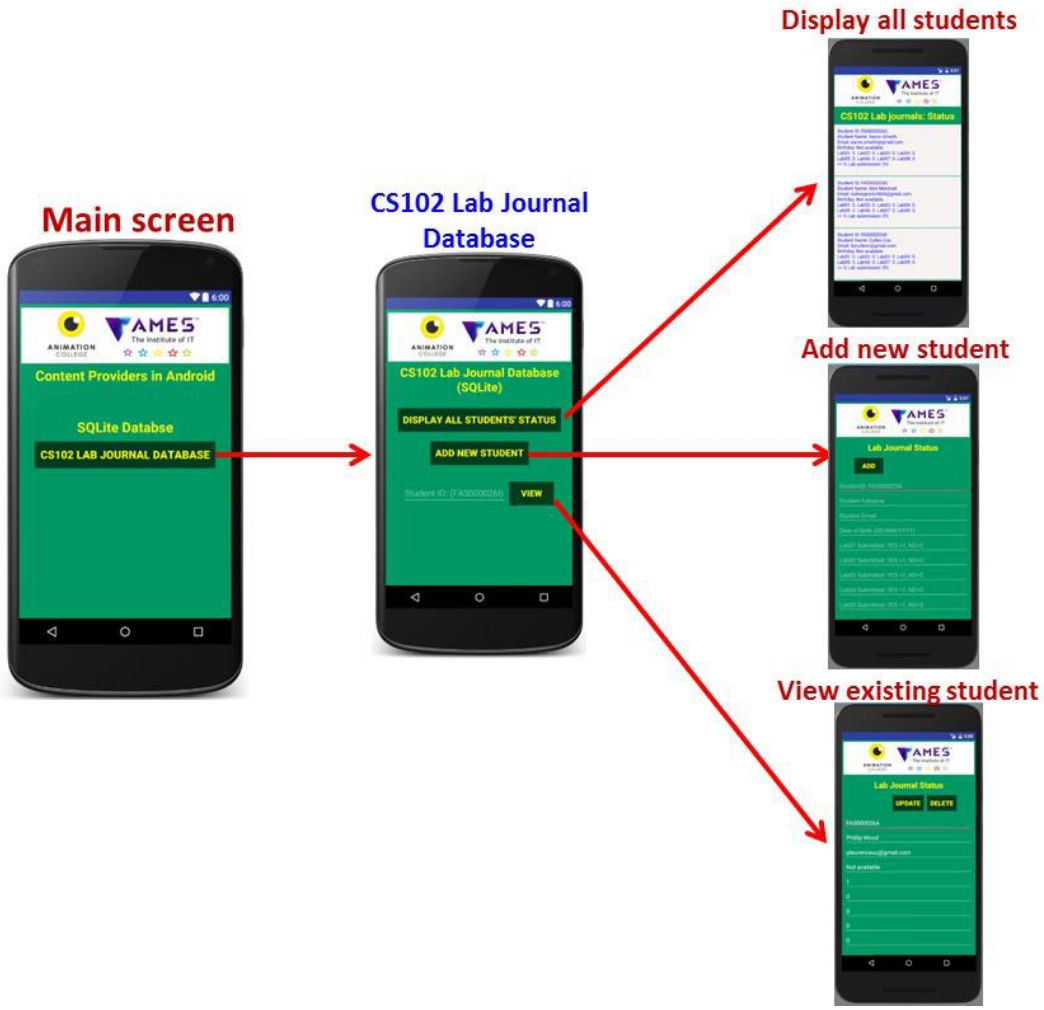
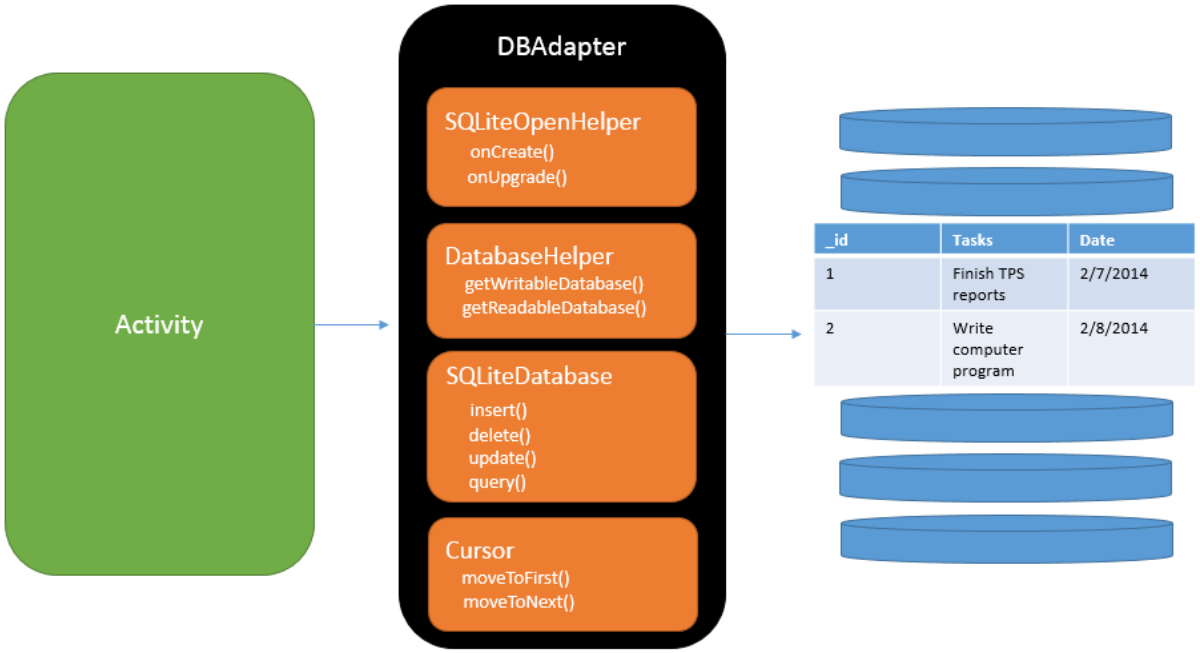


Figure 1: App wireframe

SQLite Database:

SQLite is an opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation. SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC, ODBC e.t.c



Database - Package

The main package is android.database.sqlite that contains the classes to manage your own databases.

Database - Creation

In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below:

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

Apart from this, there are other functions available in the database package , that does this job. They are listed below:

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler):** This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags):** It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	**openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory):** It not only opens but create the database if it not exists. This method is equivalent to openDatabase method
4	**openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory):** This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

Database - Insertion

We can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below:

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");
```

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs):** This method not only insert data , but also used to update or modify already existing data in database using bind arguments

Database - Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

```
Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(1);
String password = resultSet.getString(2);
```

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount():** This method return the total number of columns of the table.

2	**getColumnIndex(String columnName):** This method returns the index number of a column by specifying the name of the column
3	**getColumnName(int columnIndex):** This method returns the name of the column by specifying the index of the column
4	**getColumnNames():** This method returns the array of all the column names of the table.
5	**getCount():** This method returns the total number of rows in the cursor
6	**getPosition():** This method returns the current position of the cursor in the table
7	**isClosed():** This method returns true if the cursor is closed and return false otherwise

Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

```
public class DBHelper extends SQLiteOpenHelper {
  public DBHelper(){
    super(context,DATABASE_NAME,null,1);
  }
  public void onCreate(SQLiteDatabase db) {}
  public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}
}
```

References:

+ Android - Content Providers: https://www.tutorialspoint.com/android/android_content_providers.htm
+ Android - SQLite Database Tutorial: https://www.tutorialspoint.com/android/android_sqlite_database.htm
+ Android SQLite Database Tutorial using Android Studio: https://github.com/mobilesiri/Android-Sqlite-Database-Tutorial/tree/master/app/src/main
+ Android SQLite database and content provider – Tutorial: http://www.vogella.com/tutorials/AndroidSQLite/article.html