**ASSIGNMENT NO.**

**Title:** A Calculator mobile application with Memory Save/Recall.

**Aim:** To design a mobile application for Calculator (+, - ,\*, /, Sin, Cos, sq.-Root) with Memory Save/Recall.

**Objective:**

* To study and implement Trigonometry functionality in android studio.
* To implement memory save/recall feature in calculator.
* To study and perform Positive and Negative testing.
* To implement software design and testing in android platform.

**Theory:**

Activities in the system are managed as an *activity stack*. When a new activity is started, it is placed on the top of the stack and becomes the running activity the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

An activity has essentially four states:

* If an activity is in the foreground of the screen (at the top of the stack), it is *active* or *running*.
* If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is *paused*. A paused activity is completely alive (it maintains all state and member information and remains attached to the window manager), but can be killed by the system in extreme low memory situations.
* If an activity is completely obscured by another activity, it is *stopped*. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.
* If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

The following diagram shows the important state paths of an Activity. The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states. The colored ovals are major states the Activity can be in.

There are three key loops you may be interested in monitoring within your activity:

* The **entire lifetime** of an activity happens between the first call to onCreate(Bundle) through to a single final call to onDestroy(). An activity will do all setup of "global" state in onCreate(), and release all remaining resources in onDestroy(). For example, if it has a thread running in the background to download data from the network, it may create that thread in onCreate() and then stop the thread in onDestroy().
* The **visible lifetime** of an activity happens between a call to [onStart](https://developer.android.com/reference/android/app/Activity.html#onStart())() until a corresponding call to [onStop](https://developer.android.com/reference/android/app/Activity.html#onStop())(). During this time the user can see th**e** activity on-screen, though it may not be in the foreground and interacting with the user. Between these two methods you can maintain resources that are needed to show the activity to the user. For example, you can register a [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver.html) in onStart() to monitor for changes that impact your UI, and unregister it in onStop() when the user no longer sees what you are displaying. The onStart() and onStop() methods can be called multiple times, as the activity becomes visible and hidden to the user.
* The **foreground lifetime** of an activity happens between a call to onResume() until a corresponding call to [onPause()](https://developer.android.com/reference/android/app/Activity.html#onPause()). During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states -- for example when the device goes to sleep, when an activity result is delivered, when a new intent is delivered -- so the code in these methods should be fairly lightweight.

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 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: (*image courtesy : android.com* )

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.

**Memory Functions of your Calculator(Descrip­tion of each button):**

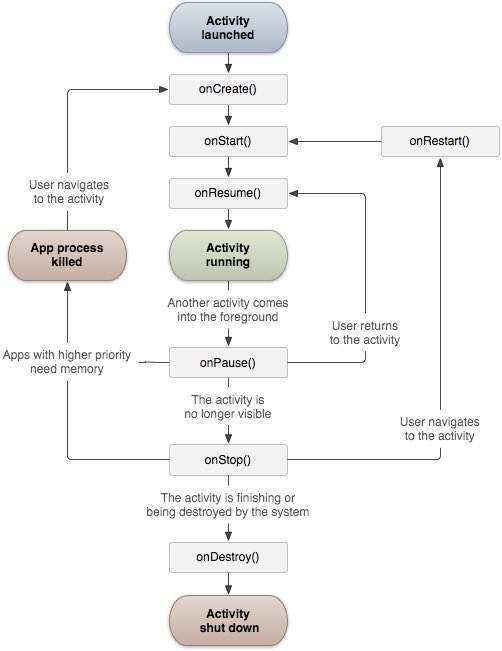
MC > Clears the memory

MR > Recall value in memory

MS > Save value into memory

M+ > Adds the cur­rently dis­played num­ber on your cal­cu­la­tor to the num­ber in memory

M– > Sub­tracts the cur­rently dis­played num­ber from the num­ber in memory



**Mathematical Model:**

Calculator Functions

**Input Process Output**

E={s, e, I, O, Fn, Sc, Fc}

Where,

s= initial state

e= end state

X= set of inputs

Y= set of outputs

F= set of function

Sc= Success cases

Fc=Failure cases

Let S’ be system in observation

Where S’ C S

S’ = {s,e,X,Y,F,Sc,Fc}

* S= start state

{expression}

* e= end state

exit(0) ….success

* X= {(i) | i € 0-9,tan,sin,cos,log}
* Y= {Y1, Y2} Є Y

Where ,

{Y1,}Є success

{ Y2} Є failure

Output O = {O1}

O1: O={Expression evaluted}

Functions Fn = {Fn1, Fn2, Fn3, Fn4, Fn5, Fn6, Fn7, Fn8}

Fn1: takeInput(){inputToBeParsed = Input };

Fn2: basic operations(){ans = a op b, op € (+,-,/,\*)};

Fn3: trigonometric operations(){tan(i),sin(i),cos(i); i€(0-360)};

Fn4: save value into memory();

Fn5: recall value from memory(){lastData = “inputToBeParsed”};

Fn6: clear memory(){inputToBeParsed = “NULL”}

Fn7: addToMemory(){Memory += “inputToBeParsed”};

Fn8: subtractFromMemory({Memory -= “ inputToBeParsed” });

Success if Sc = {Sc1, Sc2, Sc3}

Sc1: Save/Recall functions work correctly

Sc2: Memory operations are implemented correctly

Sc3: App is deployed successfully

Failure if Fc = {Fc1, Fc2}

Fc1: Failure in deployment of app

Fc2: Memory operations failure

**Use case Diagram:**

The use case diagram of the system consist of two actors, User and Calculator. The user performs functions like Giving input to the system, Pressing calculate button, Press clear button. The Calculator is responsible for performing functions like Evaluate expressions, Save the result, Display the result. The Evaluate expressions functions include Evaluating Arithmetic exp and Trigonometric exp.

**System testing:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sr. No. | Test Steps | Test Data | Expected Result | Actual Result | Status |
| 1. | Check whether all keys are working | (0-9), C, AC, DEL, tan, sin, cos. | Keys should work properly | All keys are working. | Pass |
| 2. | Check Arithmetic operations | a,b € (0-9),  op € (+, -, \*, /) | All the operation should follow bodmas rule | Operations are working correctly. | Pass |
| 3. | Check if the data is stored. | //Last Result | The last Data should be stored. | The Data is stored in the memory. | Pass. |
| 4. | Check if data is stored. | //Last Result | The last Data should be stored. | Garbage data is being stored. | Fail |
| 5. | Check whether all buttons are working | (0-9), C, AC, DEL, tan, sin, cos. | Keys should work properly | Application crashes | Fail |

**Steps**:

1. Take input from user
2. Take input operation
3. Write function to evaluate the expression
4. Add a button to save the result in memory
5. Add a button to recall the last saved result from memory
6. Addition and Subtraction of numbers from Value stored in memory.
7. Add Clear memory function

**Input:** Expression to be evaluated

**Output:** Evaluated expression

**Platform:** Windows 7, Android Studio

**Language:** Java, XML

**Conclusion:**

Thus, we have successfully completed Android Mobile App for Calculator having save/recall functionality and Trigonometry functionality and tested on android mobile.

**FAQs:**

**1. What is Intent? Explain explicit Intent and implicit Intent.**

**Ans:** An intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background Service. An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

The primary pieces of information in an intent are:

* action -- The general action to be performed, such as ACTION\_VIEW, ACTION\_EDIT, ACTION\_MAIN, etc.
* data -- The data to operate on, such as a person record in the contacts database, expressed as a Uri.

There are two primary forms of intents:

* Explicit Intents have specified a component (via setComponent(ComponentName) or setClass(Context, Class)), which provides the exact class to be run. Often these will not include any other information, simply being a way for an application to launch various internal activities it has as the user interacts with the application.
* Implicit Intents have not specified a component; instead, they must include enough information for the system to determine which of the available components is best to run for that intent.

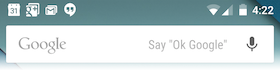
**2. What is content provider in Android?**

**Ans:** A content provider manages access to a central repository of data. A provider is part of an Android application, which often provides its own UI for working with the data. However, content providers are primarily intended to be used by other applications, which access the provider using a provider client object. Together, providers and provider clients offer a consistent, standard interface to data that also handles inter-process communication and secure data access.

Typically you work with content providers in one of two scenarios; you may want to implement code to access an exiting content provider in another application, or you may want to create a new content provider in your application to share data with other applications. This topic covers the basics of working with existing content providers.

**3. What are the notifications available in Android?**

**Ans:** A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.



Different notifications available in Android are:

* Schedule notification
* Simple notification
* Ping notification
* Inbox notification
* Big Picture notification
* Big Text notification

**4. What is ADB in Android?**

**Ans:** ADB, Android Debug Bridge, is a command-line utility included with Google’s Android SDK. ADB can control your device over USB from a computer, copy files back and forth, install and uninstall apps, run shell commands, and more.

Aside from the Android SDK, which the Android debug bridge is a part of, the basic requirements of an Android development setup is a computer that passes the minimum system requirements for running the Android SDK and, in most cases, an Android device itself. In software development lingo, the computer mentioned is known as the development machine.

The client component of the Android Debug Bridge runs in the development machine. It can be invoked from the command prompt (a.k.a. shell) using the adb command. There are also other tools such as the ADT (Android Development Tools) plugin and DDMS (Dalvik Debug Monitor Service) that can create adb clients. The ADB daemon, on the other hand, runs as a background process in either an emulator instance or in the device itself.

Finally, the server component of the ADB, which also runs in the development machine but only in the background, takes charge of managing communication between the ADB client and the ADB daemon. When the Android Debug Bridge is active, the user can issue adb commands to interact with one or more emulator instances.

The adb can also run multiple instances of the adb client, which can all be used to control all existing emulator instances. The easiest way to use the Android Debug Bridge is by installing ADT plugin into the Eclipse IDE (Integrated Development Environment). This way, the developer wouldn’t have to enter commands through command prompt.