As an android developer, if one does not know the application lifecycle of android application or does not have in-depth knowledge about it, there are very high chances that the application will not have a good user experience. Not having proper knowledge of the application lifecycle will not affect the working of that application but it will lead to poor user experience. Let's take one example where the user is using the application, which takes a high amount of battery usage and on the other hand there is one another same application providing the same functionalities offered by the previous app, then there are very high chances that the user will shift to later application and will not be using the previous app. Having a lot of battery usage by the app can be the indication that the android developer of that app does not have proper knowledge of the application lifecycle.

This is an important topic and mandatory need to have proper knowledge of the processes in the android and application lifecycle and how different processes can affect the application state. The knowledge of the Application Lifecycle in Android is also a must because without having proper knowledge many times the process can be crashed while doing some important work which will ultimately lead to a bad impression of the app on the users. The lifecycle of the application's process is not solely controlled by the application itself, but the application's process lifetime depends on the various numbers of factors such as memory taken by the application, memory available in the system, how important is the process from the perspective of the user, how long it had been since the user not used that application.

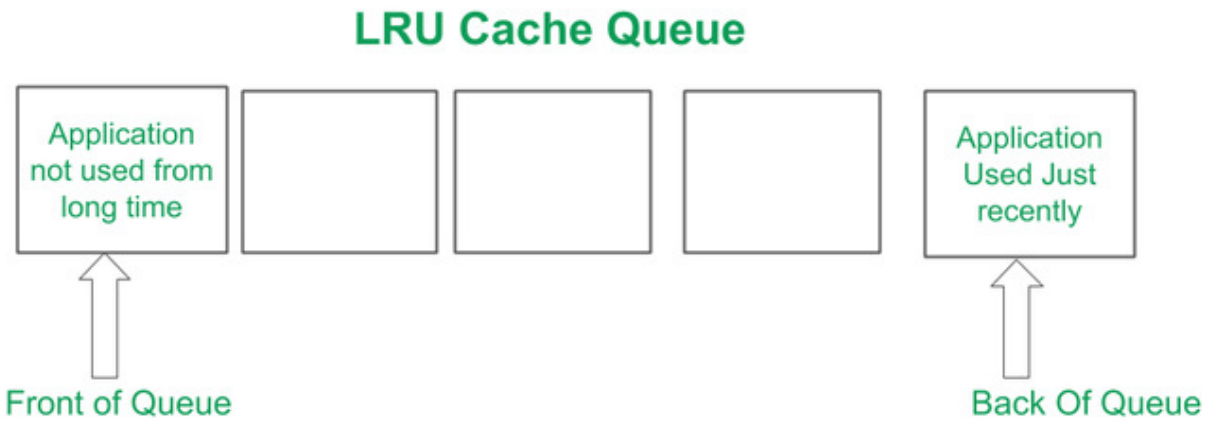
Importance Of Having Knowledge of Process and Application Lifecycle

Knowing the Application lifecycle is necessary as though it will not affect the working of the app, but it will affect the user experience. Let say the user has two option in which user has the choice to choose one app of out two, in which both the application offers the same functionality, and involving the same type of task, but one app consumes more battery usage than the other, so, surely, the user will go for the application which consumes less battery. Thus developers need to understand the application is not holding resources and services when it is not in use, as it can be the cause of more battery usage. Thus developers should handle each application state of the android app and should release the resources when the app is in the background state.

LRU Cache Role in Optimization Of Android Applications

As it is known that a particular android device has limited storage capacity and processing speed, but since there are so many applications running on the user device, it becomes challenging for the android operating system to manage the resource properly so that none of the resources become a deficit of resource. So android operating system pushes the least used application in a cache called as LRU (Least Recently Used) Cache, if the application has not been used for the very long time, it will be pushed to the queue of the LRU and will be present at the front of the queue. If the application comes into use in the near time, it will be near to the back of the queue, and if the application is most recently used it will be present at the back of the queue. For example, if the user has not used Facebook for a very long time, then the Facebook application will be present at the front of the queue and if let's suppose the

user has used WhatsApp recently, then it will be present at the back of the LRU Cache queue. The LRU Cache Queue can be diagrammatically represented as:



The Priority of Processes In Android Application

To determine which process should be killed to proper memory and battery management, the Android operating system maintains a hierarchy in which all the processes are placed in order of their priority. The less priority is the process which can be killed first when the system wants to free up some resource. Android uses some set of rules and regulations to decide the priority of the processes based on the running state of the applications. Below are the process states that a process may have at any time in android applications. The priority of these processes decreases from top to down in order in which they are listed.

1. Foreground process

A foreground process is a process with which the user is currently interacting and using it. A process is considered to be in the foreground state if any of the below conditions hold:

If the process is running an activity with which the user is interacting

If it has a broadcast receiver which is currently in execution to receive any system update.

Example: Imagine the user is using Whatsapp, so the Whatsapp app will be said to be in the foreground state. This process is of the highest priority and they can only be killed by the system if the memory is so low that even this process cannot continue their execution.

2. Visible process

A visible process is a process when the activity can be visible to the user. The user does not directly interact with this process, as the activity corresponds to this process would be covered partially by another activity and the process will be in the onPause() lifecycle state. This process cannot be killed unless there is so much lack of memory in the system that the execution of these processes cannot be possible. Killing these processes will create a negative impact on user experience, as a user can see the

activity corresponding to this process. These processes would be killed only when keeping them alive make it impossible for the foreground process to continue their execution.

Example: When some application needs permission like camera access, storage access, etc a prompt or dialog box will appear and ask for the required permission. So at this time, the process corresponding to the activity of the app which is running previously will go in the visible state.

3. Service Process

A process is said to be a service process if it is in running state and neither a foreground process and a visible process. These processes are not directly visible to the user of the application. This process is helpful for the applications which perform the background tasks such as background network data upload or download. The system will keep the service process alive until it becomes impossible for the system to keep the foreground process and visible process running.

Example: Uploading a PDF on the Whatsapp from the desktop is a service process that is done in the background.

4. Background process

A background state in which the onStop() lifecycle method of android is called by the system. Let's suppose the user is using an app and suddenly presses the home button, so because of this action, the process goes from foreground state to background state. When the app goes from foreground state to background state, it goes to the LRU cache queue and will be placed in the front of the queue.when the user returns to that app, the process will return from background state to foreground state. So having knowledge of the process and application lifecycle in android along with how processes can decide the lifecycle time of the application is a must for an android developer which can lead to good user experience.

5. Empty process:

A process is said to be in the empty state if it doesn't come under the category of the above four mentioned process states. In an empty process, there is no active component of the application i.e. each and every component of the process will the in stop state. The application can be put in the LRU for better caching purpose but is memory is not present or low in amount, then that application will be removed from the cache also.

Battery Usage Of Phone When Using the Same Type Of Applications

From the above image, it can be said that applications that offer the same type of services lead to near about the same amount of battery consumption. (Consider the usage time of application to be equal) From the above image, it can be seen that as Youtube and Google meets are both Video Streaming and Linkedin and Whatsapp are both the same kinds of messaging app, though a little bit different, but these apps use the same kind of data-transport and services, thus consumes the same amount of battery.

Android GUI Architecture

The Android environment adds yet another Graphical User Interface (GUI) toolkit to the Java ecosphere, joining AWT, Swing, SWT, and J2ME (leaving aside the web UI toolkits). If you've worked with any of these, the Android framework will look familiar. Like them, it is single-threaded, event-driven, and built on a library of nestable components.

The Android UI framework is, like the other UI frameworks, organized around the common Model-View-Controller pattern illustrated below. It provides structure and tools for building a Controller that handles user input (like key presses and screen taps) and a View that renders graphical information to the screen.

The Model

The Model is the guts of your application: what it actually does. It might be, for instance, the database of tunes on your device and the code for playing them. It might be your list of contacts and the code that places phone calls or sends IMs to them.

While a particular application's View and Controller will necessarily reflect the Model they manipulate, a single Model might be used by several different applications. Think, for instance, of an MP3 player and an application that converts MP3 files into WAV files. For both applications, the Model includes the MP3 file format and codecs for it. The former application, however, has the familiar Stop, Start, and Pause controls, and plays the track. The latter may not produce any sound at all; instead, it will have controls for setting bitrate, etc. The Model is all about the data. It is the subject of most of the rest of this book.

The View

The View is the application's feedback to the user. It is the portion of the application responsible for rendering the display, sending audio to speakers, generating tactile feedback, and so on. The graphical portion of the Android UI framework's View, described in detail in Chapter 12, is implemented as a tree of subclasses of the View class. Graphically, each of these objects represents a rectangular area on the screen that is completely within the rectangular area represented by its parent in the tree. The root of this tree is the application window.

As an example, the display in a hypothetical MP3 player might contain a component that shows the album cover for the currently playing tune. Another component might display the name of the currently playing song, while a third contains subcomponents such as the Play, Pause, and Stop buttons.

The UI framework paints the screen by walking the View tree, asking each component to draw itself in a preorder traversal. In other words, each component draws itself and then asks each of its children to do the same. When the whole tree has been rendered, the smaller, nested components that are the leaves of the tree—and that were, therefore, painted later—appear to be painted on top of the components that are nearer to the root and that were painted first.

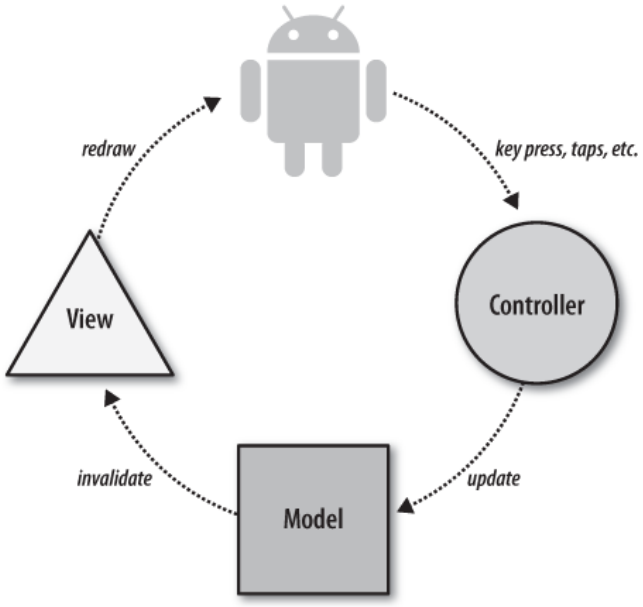
The Android UI framework is actually quite a bit more efficient than this oversimplified description suggests. It does not paint an area of a parent view if it can be certain that some child will later paint the same area, because it would be a waste of time to paint background underneath an opaque object! It would also be a waste of time to repaint portions of a view that have not changed.

The Controller

The Controller is the portion of an application that responds to external actions: a keystroke, a screen tap, an incoming call, etc. It is implemented as an event queue. Each external action is represented as a unique event in the queue. The framework removes each event from the queue in order and dispatches it.

For example, when a user presses a key on his phone, the Android system generates a KeyEvent and adds it to an event queue. Eventually, after previously enqueued events have been processed, the KeyEvent is removed from the queue and passed as the parameter of a call to the dispatchKeyEvent method of the View that is currently selected.

Once an event is dispatched to the in-focus component, that component may take appropriate action to change the internal state of the program. In an MP3 player application, for instance, when the user taps a Play/Pause button on the screen and the event is dispatched to that button's object, the handler method might update the Model to resume playing some previously selected tune.



Android Layout and Views

I hope you are enjoying the android journey so far and are developing some awesome apps. Today, you will get to know a pretty exciting topic known as Layouts and Views. You are very well aware that an interactive UI attracts more users to your application. To build such an interactive user interface, you should have a crisp knowledge of Layouts, Views, and View Groups.

So, through this article, you will come across the several layouts, view, and view groups present in android.

What is an Android Layout?

A layout defines the structure of your application. In other words, a layout is a container that holds several view elements inside it. The view elements are TextView, ImageView, Button, EditText, etc. Layouts have their params and using them, we can alter the margins, paddings, etc., for a given layout.

Whenever you develop an application, you first need to choose the layout and style it accordingly to make your application look better. Every Layout in android is styled using the XML language, as shown below.

Code:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   tools:context=".MainActivity">

   <TextView
      android:id="@+id/title"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
```

```
    android:text="TechVidvan Android"
    android:textSize="40sp"
    android:textColor="#078A0D"
    android:textStyle="bold"
    android:gravity="center"
    android:layout_margin="20dp"
    />


</LinearLayout>
```

Types of Layouts in Android

Android provides you with several types of layouts depending upon your needs. Let's see all of them with a brief description of each.

Layout Type	Description
Linear Layout	Linear Layout is used to align your items in a linear fashion, either vertical or horizontal.
Relative Layout	Relative Layout is used when you wish to place views relative to each other's position.
Table Layout	Table Layout is used to display the items(such as text, image, etc) in the form of rows and columns.
Absolute Layout	Absolute Layout is used when you want to fix the position of an element in the screen.
Frame Layout	Frame Layout is used when you deal with fragments. This layout allows you only to have one child view.
List View	List View is used when you wish to display your items in the form of a list.
Grid View	Grid View is used when you wish to show your data in the form of a grid of one's, two's, three's, or so on.
Web View	Web View is a layout that is used mainly when dealing with web pages inside the application.

Scroll View Scroll View allows your content to scroll vertically or horizontally. The point to note is that ScrollView has only one child view.

Constraint Layout Constraint layout is the most adaptive layout among all the above. It allows you to put vertical, horizontal, top, and bottom constraints for any element.

Android Layout Attributes

Now let's see the attributes you can apply to layouts to style them and provide more functionalities.

Attribute Name Description

android:id The "id" attribute helps in the unique identification of the layout in your design.

android:layout_width The "layout_width" attribute sets the width of the layout.

android:layout_height The "layout_height" attribute sets the height of the layout.

android:layout_marginTop The "layout_marginTop" attribute specifies the margin of the layout from the top.

android:layout_marginBottom The "layout_marginTop" attribute specifies the margin of the layout from the bottom.

android:layout_marginLeft The "layout_marginTop" attribute specifies the margin of the layout from the left side.

android:layout_marginRight The "layout_marginTop" attribute specifies the margin of the layout from the right side.

android:layout_gravity The "layout_gravity" attribute specifies the position of the layout on the screen. The possible positions are center, top, bottom, left, right, etc.

android:layout_weight The "layout_weight" attribute is usually applied to the views inside the layout. It specifies how many portions(or free space) of layout that view should occupy.

android:layout_x The "layout_x" attribute is used to specify the x coordinates for your layout.

android:layout_y The "layout_y" attribute is used to specify the y coordinates for your layout.

android:paddingLeft The "paddingLeft" attribute is used to provide padding to the layout from the left.

android:paddingRight The "paddingLeft" attribute is used to provide padding to the layout from the right.

android:paddingTop The “paddingTop” attribute is used to provide padding to the layout from the top.

android:paddingBottomThe “paddingBottom” attribute is used to provide padding to the layout from the bottom.

What is an Android View?

A View is a fundamental element for any user interface (or design) in android. View provides you with a rectangular space that can be used to display, interact or handle events. The view is a superclass for any graphical components in android. Usually, each view is rectangular and can hold textual and visual contents. Attached to a view, there are several attributes that define the property and functionality of that view.

Below is a general example of defining a view using XML.

Code:

```
<YourViewName
   Attribute_1 = value_1
   Attribute_2 = value_2
   Attribute_3 = value_3
   Attribute_4 = value_4
   .
   .
   .
   Attribute_n = value_n
/>
```

Views, as you know, are the elements that you can add to your user interface. Views are constructive in creating fabulous, incredible, and gorgeous user interfaces. Along with this, the views are used to make your interface interactive and available supporting clicks, gestures, touches, etc.

Types of Android Views

Android provides a bunch of views that can be used depending upon the needs of your application. Some of them are as follows:

TextView

ImageView

EditText

Button

ImageButton

ToggleButton

RadioButton

RadioGroup

CheckBox

What is Android ViewGroup?

Android ViewGroup is a container that holds several views, layouts, or even other view groups inside it. Using the view group, you can give rise to several views and layouts. Android ViewGroup is a superclass for all layouts and acts as a subclass for the View class. ViewGroup can be thought of as an exceptional view known as parent view, and the views inside it are known as child views.

Some of the most commonly used ViewGroups are as follows:

FrameLayout

WebView

LinearLayout

RelativeLayout

TableLayout

Constraint Layout

ListView

GridView

View Identification in Android

View identification is essential in android because you can’t respond to the user without identifying your view elements. You should locate the view to know which view was triggered by the user and what response you should provide.

So, first of all, you need to provide an id to your view element, and then in your Kotlin class, you need to use the findViewByID() method to bind the view element with a Kotlin class variable.

XML Code:

```
<TextView
    android:id="@+id/title"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="TechVidvan Android"
    android:textSize="40sp"
    android:textColor="#078A0D"
    android:textStyle="bold"
    android:gravity="center"
    android:layout_margin="20dp"
    />
```

Difference between Android View and Android ViewGroup

View	ViewGroup
The view is a fundamental UI component that responds to user interactions(touches, clicks, scrolls, etc.).	ViewGroup is a container that holds several views, layouts, and view groups inside it.
The view is a superclass for many view elements like TextView, ImageView, Button, etc.	ViewGroup is a superclass for various layouts like LinearLayout, RelativeLayout, etc.
The view belongs to the **android.view.View class**	ViewGroup belongs to the **android.view.ViewGroup class**
A View object is usually a widget, for example, Button, TextView, ImageView, etc.	A ViewGroup object is typically a layout containing several views and layouts inside it.
View, in general, is known as a **child view**.	ViewGroup is known as **parent view** holding several child views.