**Assignment-2**

1. **Draw and Explain Android Activity Application Life Cycle?**

|  |
| --- |
| Activity lifecycle |

* Lifecycle is one of the [Android Architecture Components](https://www.geeksforgeeks.org/jetpack-architecture-components-in-android/) which was released by Google to make it easier for all the Android developers. The **Lifecycle** is a class/interface which holds the information about the state of an activity/fragment and also it allows other objects to observe this state by keeping track of it. The LifeCycle component is concerned with the Android LifeCycle events of a component such as an [Activity](https://www.geeksforgeeks.org/activity-lifecycle-in-android-with-demo-app/)or a [Fragment](https://www.geeksforgeeks.org/fragment-lifecycle-in-android/). Lifecycle is a process that tells us about the**Events** performed on an Activity/Fragment. We have a lifecycle as a class that has two types of enumerations to track the components, State and Event. Event and State are used to determine the lifecycle. Each event has its own state. To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: [onCreate()](https://developer.android.com/reference/android/app/Activity#onCreate(android.os.Bundle)), [onStart()](https://developer.android.com/reference/android/app/Activity#onStart()), [onResume()](https://developer.android.com/reference/android/app/Activity#onResume()), [onPause()](https://developer.android.com/reference/android/app/Activity#onPause()), [onStop()](https://developer.android.com/reference/android/app/Activity#onStop()), and [onDestroy()](https://developer.android.com/reference/android/app/Activity" \l "onDestroy()). The system invokes each of these callbacks as the activity enters a new state.

|  |  |
| --- | --- |
| **Event** | **State** |
| **OnCreate()** | Called when the activity is first created. |
| **OnStart()** | Called when the activity becomes visible to the user. |
| **OnResume()** | Called when the activity starts interacting with the user. |
| **OnPause()** | Called when the current activity is being paused and the previous activity is resumed. |
| **OnStop()** | Called when the activity is no longer visible to the user. |
| **OnDestroy()** | Called before the activity is destroyed by the system(either manually or by the system to conserve memory |
| **OnRestart()** | Called when the activity has been stopped and is restarting again. |

1. **Explain Layout and Different Type of Layout?**

* Android **Layout** is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen.
* **Types of Android Layout**
* **Android Linear Layout:**LinearLayout is a ViewGroup subclass, used to provide child View elements one by one either in a particular direction either **horizontally or vertically** based on the orientation property.
* **Android Relative Layout:**RelativeLayout is a ViewGroup subclass, used to specify the position of child View elements relative to each other like (A to the right of B) or relative to the parent (fix to the top of the parent).
* **Android Constraint Layout:**ConstraintLayout is a ViewGroup subclass, used to specify the position of layout constraints for every child View relative to other views present. A ConstraintLayout is similar to a RelativeLayout, but having more power.
* **Android Frame Layout:**FrameLayout is a ViewGroup subclass, used to specify the position of View elements it contains on the top of each other to display only a single View inside the FrameLayout.
* **Android Table Layout:**TableLayout is a ViewGroup subclass, used to display the child View elements in rows and columns.
* **Android Web View:**WebView is a browser that is used to display the web pages in our activity layout.
* **Android ListView:**ListView is a ViewGroup, used to display scrollable lists of items in a single column.
* **Android Grid View:**GridView is a ViewGroup that is used to display a scrollable list of items in a grid view of rows and columns.

1. **Application Priority and process states?**

* **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.

1. **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 inthe **onPause()**[lifecycle state](https://www.geeksforgeeks.org/activity-lifecycle-in-android-with-demo-app/). 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.

1. **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.

1. **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.
2. **Difference between Linear Layout and Relative layout?**

|  |  |
| --- | --- |
| **LinearLayout** | **RelativeLayout** |
| We can adjust views and widgets linearly i.e. Horizontally and vertically. | We can adjust views and widgets according to one’s satisfaction. |
| **layout\_weight** attribute in the linear layout is used to specify the equal or specific size to the particular widget and view by using the following attribute. **android:layout\_weight = ‘0’**Here Weight is specified as 0 in order to give equal size or space to each view or widget. | Various attributes like: layout\_toRightOf, layout\_toLeftOf, layout\_below, layout\_alignParentTop, layout\_top, layout\_alignParentLeft, layout\_alignParentRight are used to specify the position of each view and widget. |
| It is  useful when we arrange views in a linear fashion | It is useful when we arrange views in a relative fashion. |
| **Syntax:**  <LinearLayout>  <!–Views, widgets–>  </LinearLayout> | **Syntax:**  <RelativeLayout>  <!–Views, Widgets–>  </RelativeLayout> |
| **Example:** In various Apps, LinearLayout is mainly applicable in the SignUp screen where Name, Email, Phone Number, Submit, etc. are arranged in a linear fashion. | **Example:**In Google Play Store, when we open the app, the games, books, movies, and App’s sections all are arranges in Relative Layout Fashion. |
| LinearLayout is less used as compared to RelativeLayout. | RelativeLayout is used more in applications. |
| We can use LinearLayout inside RelativeLayout. | We can also use RelativeLayout as a Child of LinearLayout. |

1. **Explain Views and View Group in Short?**

|  |
| --- |
| Introduction to Android Views | Studytonight |

* The View class is the base class or we can say that it is the superclass for all the GUI components in android. For example, the EditText class is used to accept the input from users in android apps, which is a subclass of View, and another example of the TextViewclass which is used to display text labels in Android apps is also a subclass of View. **View**refer to the **android.view.View** class, which is the base class of all UI classes. android.view.View class is the root of the UI class hierarchy. So from an object point of view, all UI objects are View objects. Following are some of the common View subclasses that will be used in android applications.
* [TextView](https://www.geeksforgeeks.org/working-with-the-textview-in-android/)
* [EditText](https://www.geeksforgeeks.org/edittext-widget-in-android-using-java-with-examples/)
* [ImageView](https://www.geeksforgeeks.org/imageview-in-android-with-example/)
* [RadioButton](https://www.geeksforgeeks.org/radiobutton-in-kotlin/)
* [Button](https://www.geeksforgeeks.org/button-in-kotlin/)
* [ImageButton](https://www.geeksforgeeks.org/imagebutton-in-kotlin/)
* [CheckBox](https://www.geeksforgeeks.org/how-to-use-checkbox-in-android/)
* [DatePicker](https://www.geeksforgeeks.org/datepicker-in-kotlin/)
* [Spinner](https://www.geeksforgeeks.org/spinner-in-android-using-java-with-example/)
* [ProgressBar](https://www.geeksforgeeks.org/progressbar-in-kotlin/)and etc.

|  |
| --- |
| https://miro.medium.com/v2/resize:fit:643/1*J8oiR9xXYZJLORODJqjP4g.jpeg |

### ****ViewGroup:**** The ViewGroup class is a subclass of the View class. And also it will act as a base class for layouts and layouts parameters.  The ViewGroup will provide an invisible container to hold other Views or ViewGroups and to define the layout properties. For example, Linear Layout is the ViewGroup that contains UI controls like Button, TextView, etc., and other layouts also. ****ViewGroup****Refer to the ****android.view. ViewGroup**** class, which is the base class of some special UI classes that can contain other View objects as children. Since ViewGroup objects are also View objects, multiple ViewGroup objects and View objects can be organized into an object tree to build a complex UI structure. Following are the commonly used ViewGroup subclasses used in android applications.

* [FrameLayout](https://www.geeksforgeeks.org/android-framelayout-in-kotlin/)
* [WebView](https://www.geeksforgeeks.org/android-webview-in-kotlin/)
* [ListView](https://www.geeksforgeeks.org/android-listview-in-java-with-example/)
* [GridView](https://www.geeksforgeeks.org/gridview-in-android-with-example/)
* [LinearLayout](https://www.geeksforgeeks.org/linearlayout-and-its-important-attributes-with-examples-in-android/)
* [RelativeLayout](https://www.geeksforgeeks.org/android-relativelayout-in-kotlin/)
* [TableLayout](https://www.geeksforgeeks.org/android-tablelayout-in-kotlin/)and many more.

The ViewGroup subclasses listed above group View instances together and takes care of their layout. For instance, the LinearLayout will render the components after each other either horizontally or vertically.

1. **What is meant by Draw able Resources?**

* A**drawable resource** is a common concept for a graphic that can be drawn to the screen and which one can retrieve with APIs such as **getDrawable(int)** or apply to another XML resource with attributes such as**android:drawable** and **android:icon**. There are several different types of drawable resources files and let’s discuss all the types in a tabular manner. As it is known to all that **Android Studio** is the official **integrated development environment** for Google’s Android operating system, built on JetBrains’ IntelliJ IDEA software and designed specifically for Android development. So as a beginner in android app development, the developer should know about the tools vividly before building some awesome projects in Android. So in this article let’s learn to create a drawable resource XML file in Android Studio. **Drawable Resource XML** is mostly created in the **drawable folder** and is used in Android to add more customization for views. Here is the step by step process to create a new Drawable Resource XML in Android Studio.

|  |  |
| --- | --- |
| **Drawable Resources Files** | **Description** |
| **Bitmap File** | A bitmap graphic file. Android supports bitmap files in three formats: **.png**, **.jpg**, **.gif**. |
| **Nine-Patch File** | A PNG file with stretchable regions to allow image resizing based on content (.9.png) |
| **State List** | An XML file that references different bitmap graphics for different states (for example, to use a different image when a button is pressed) |
| **Layer List** | A Drawable that manages an array of other Drawables. These are drawn in array order, so the element with the largest index is be drawn on top. |
| **Level List** | An XML file that defines a drawable that manages a number of alternate Drawables, each assigned a maximum numerical value. |
| **Transition Drawable** | An XML file that defines a drawable that can cross-fade between two drawable resources. |
| **Clip Drawable** | An XML file that defines a drawable that clips another Drawable based on this Drawable’s current level value. |
| **Inset Drawable** | An XML file that defines a drawable that insets another drawable by a specified distance. This is useful when a View needs a background drawable that is smaller than the View’s actual bounds. |
| **Scale Drawable** | An XML file that defines a drawable that changes the size of another Drawable based on its current level value. |
| **Shape Drawable** | An XML file that defines a geometric shape, including colors and gradients. |

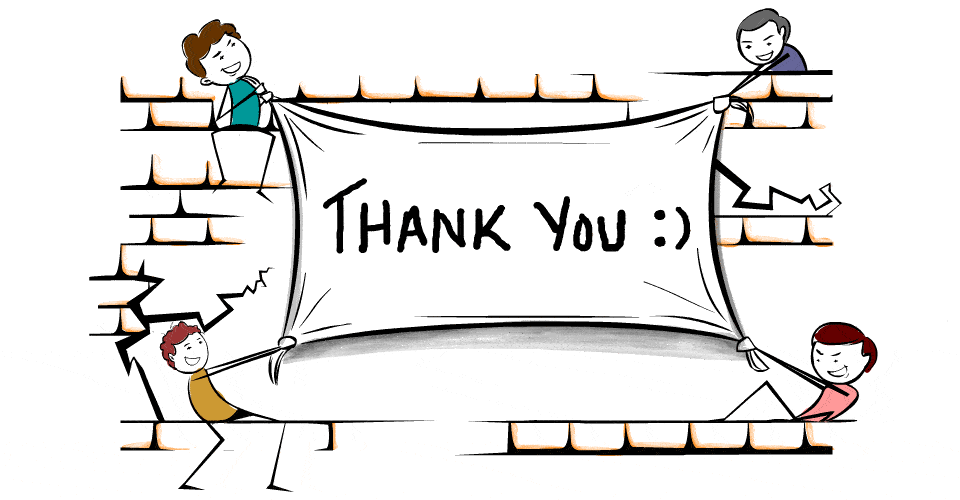
1. **Draw and Explain Fragments lifecycle?**

* A fragment has its own layout and its own behaviour with its own life cycle callbacks. We can add or remove fragments in an activity while the activity is running. We can combine multiple fragments in a single activity to build a multi-pane UI. A fragment can be used in multiple activities. Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped. A fragment can implement a behaviour that has no user interface component. Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.

|  |
| --- |
| Fragment |

* **Here is the list of methods which you can to override in your fragment class −**
* **onAttach():** The fragment instance is associated with an activity instance.The fragment and the activity is not fully initialized. Typically you get in this method a reference to the activity which uses the fragment for further initialization work.
* **onCreate():** The system calls this method when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
* **onCreateView():** The system calls this callback when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a **View** component from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
* **onActivityCreated():** The onActivityCreated() is called after the onCreateView() method when the host activity is created. Activity and fragment instance have been created as well as the view hierarchy of the activity. At this point, view can be accessed with the findViewById() method. example. In this method you can instantiate objects which require a Context object
* **onStart():** The onStart() method is called once the fragment gets visible.
* **onResume():** Fragment becomes active.
* **onPause():** The system calls this method as the first indication that the user is leaving the fragment. This is usually where you should commit any changes that should be persisted beyond the current user session.
* **onStop():** Fragment going to be stopped by calling onStop()
* **onDestroyView():** Fragment view will destroy after call this method
* **onDestroy():** onDestroy() called to do final clean up of the fragment's state but Not guaranteed to be called by the Android platform.
* **Reference:** <https://www.tutorialspoint.com/android/android_fragments.htm>

**(PPT Give you more understanding than PDF)** The material for the PDF has been compiled from various sources such as books, tutorials (offline and online), lecture notes, several resources available on Internet. The information contained in this PDF is for general information and education purpose only. While we endeavor to keep the information up to date and correct, **we make no representation of any kind about the completeness and accuracy of the material.** The information shared through this PDF material should be used for educational purpose only.

****