ANDROID

SESSION SIX

Agenda

- Events
- Event Handling
- Multiple Activities and Intents
- Intents
- Save and Restore State

MAKING THE APP INTERACTIVE

Making the App Interactive

Creating an engaging and interactive Android app goes beyond just visual design—it’s about ensuring smooth navigation, responsive feedback, and seamless transitions between multiple activities. A well-structured multi-activity app should provide intuitive user flows, dynamic UI updates, and engaging interactions that keep users immersed in the experience.

Dealing With The Views

Android development, you can create and manage views in two ways:

- Statically (XML-Based UI) :
  - This approach involves defining the UI layout using XML files. It is commonly used for designing static interfaces where the structure is predefined.
- Dynamically (Programmaticlly)
  - This approach creates and modifies views at runtime using code. It is useful when UI elements need to be generated dynamically based on user interactions or data changes.

Modify the View Dynamically

- Get The The Reference To The View in the View Hierarchy.

```
val  welcomeText : TextView = findViewById(R.id.welcomeText)
```

- Change Properties or call the methods for the View Instance

```
welcomeText.text="Ash-Shimaa weclome U All"

welcomeText.setTextColor(Color.RED)
```

Events

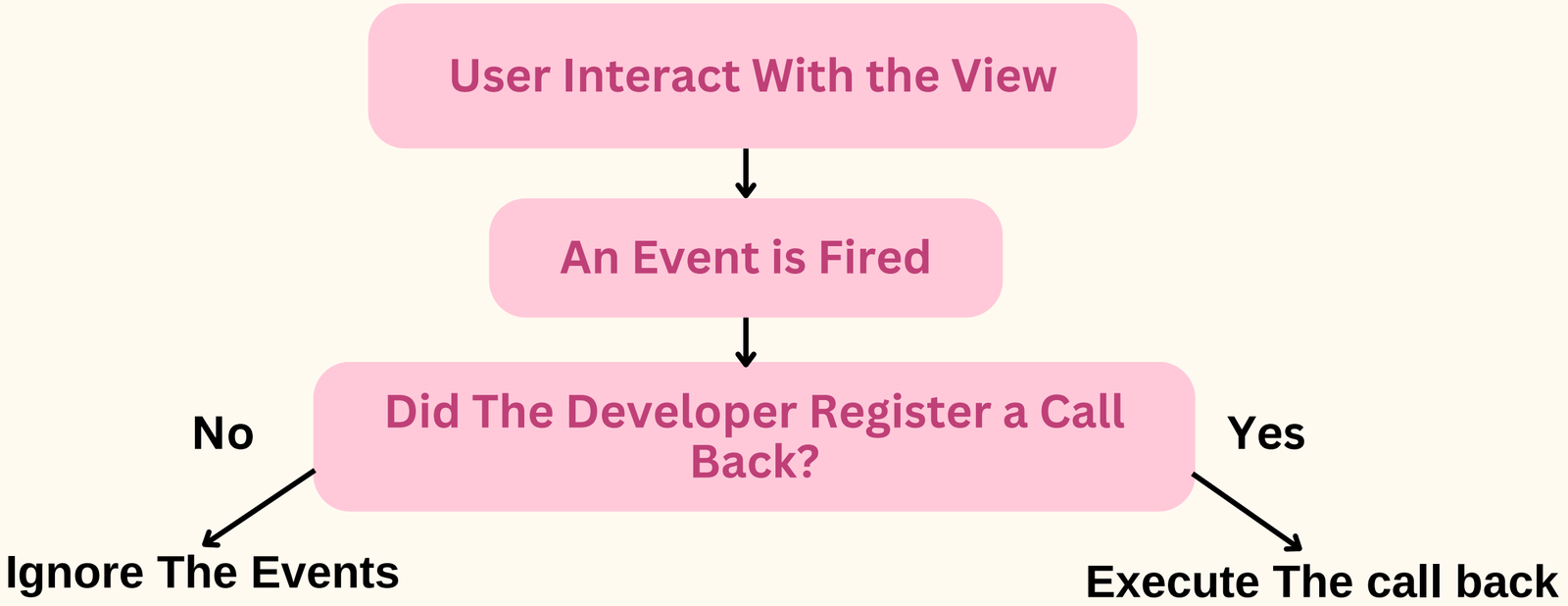
In Android, events are actions triggered by user interactions or system changes. You can handle events in various ways to make your app more interactive.

Examples ：

- Click Events
- Hover Events
- Drag

Event Handling

- Event handlers are used to respond to various user actions like clicks
- These handlers can be **set up to listen for specific actions** (e.g., click, drag) and perform tasks when those events are triggered.



Practice On Events

- 2 - Buttons (Show and Hide)
- TextView
- When :
  - Click On Button (Show) the text appear.
  - Click On Button (Hide) the text disappear.

Let’s Try It...

MULTIPLE ACTIVITIES AND INTENTS

Multi-App Screens

When building an app with multiple screens, the functionality is often divided to provide a more organized and user-friendly experience. Some examples of how different screens can be used in an app include:

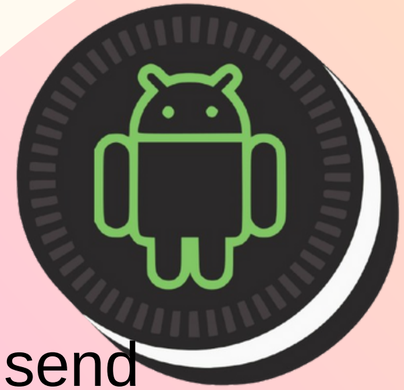
Here are the screen names for the same app :

- Home Screen.
- Search Screen .
- Checkout Screen.
- Settings Screen .
- Help/Support Screen.
- Product Detail Screen.
- Shopping Cart Screen.
- User Profile Screen.
- Order Confirmation Screen.

Intent

- Is a Description of an operation to be performed
- A facility for late run-time binding between components.
- Can launch a component in the same or a different application.
- A passive data structure holding an abstract description of an operation to be performed.
- An Intent is an object used to request an action from another app component via the Android system.

What Intents Can Do ?



- **Launch components**: They can start activities, services, or send broadcasts to other components, either within the same app or in other apps.
- **Pass data**: Intents can carry data in the form of key-value pairs, allowing data to be shared between components.
- **Request operations**: Intents describe an operation, such as viewing a website, sending an email, or opening the camera app.

Common Uses Of Intents :

- Starting a new Activity: To navigate between different screens in your app Or in another App .
- Starting a Service: To perform background tasks (like downloading data or playing music).
- Sending Data: Passing data (e.g., strings, numbers) between components.
- Broadcasting Messages: To notify other components or apps of a system-wide event (like battery low or Wi-Fi status change).

Intent Types

1- Explicit Intents :

An explicit intent specifies the target component directly by using the component's class name “ **The Same App** “

```
// Starting an activity within the same app
val intent = Intent(this, TargetActivity::class.java)
startActivity(intent)
```

Let’s Try It...

Explicit Intent Cont’d

To Send Data To The started Activity Use **putExtra**

```
intent.putExtra("username", "JohnDoe")
intent.putExtra("age", 30)
startActivity(intent)
```

To Receive It Use **getString/IntExtra**

```
val extras = intent.extras
extras?.let {
    val username = it.getString("username")
    val age = it.getInt("age")
    // Use the retrieved data
}
```

Let’s Try It...

Explicit Intent Cont’d

Another App Navigation:

```
fun openExternalApp() {
    val intent = Intent("com.example.workapp.FILE_OPEN")
    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    }
}
```

Note : **You Should Konw The package Name You’re Going To.**

Intent Types

2-Implicit Intent

- An Intent usually has two primary pieces of information:
    - Action to be performed (for example ACTION_VIEW,ACTION_EDIT, ACTION_MAIN) .
    - Data to operate on (for example, a person's record in the contacts database) .

Note : Commonly used to specify a request to transition to another Activity.

Intent Types

2-Implicit Intent

Does not specify a particular component by name. Instead, it declares a general action to perform, allowing the Android system to determine the most appropriate component (from any app) to handle the request.

1. Create an Intent ➔ `val intent = Intent(action, uri)`
2. Use the Intent to start the Activity ➔ `startActivity(intent)`

Actions :

- ACTION_VIEW
- ACTION_SEND
- ACTION_DIAL

Implicit Intent Cont'd

ACTION_VIEW :

```
val uri = Uri.parse("http://www.google.com")
val intent = Intent(Intent.ACTION_VIEW, uri)
startActivity(intent)
```

ACTION_DIAL :

```
val uri = Uri.parse("tel:8005551234")
val intent = Intent(Intent.ACTION_DIAL, uri)
startActivity(intent)
```

Sending and Retrieving Data

Sending Email :

```
fun sendEmail() {
    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "text/plain"
    intent.putExtra(Intent.EXTRA_EMAIL, emailAddresses)
    intent.putExtra(Intent.EXTRA_TEXT, "How are you?")

    if (intent.resolveActivity(packageManager) != null) {
        startActivity(intent)
    }
}
```

Sending and Retrieving Data

In the first (sending) Activity:

1. Create the Intent object
2. Put data or extras into that Intent
3. Start the new Activity with startActivity()

In the second (receiving) Activity:

1. Get the Intent object, the Activity was started with Retrieve the data or extras from the Intent object

Sending and Retrieving Data

2- Types Of Sending Data With Intents :

- **Data**: one piece of information whose data location can be represented by a URI Intent
    - // **A web page URL**

      intent.data = Uri.parse("http://www.google.com")
    - // **a Sample file URI**

      intent.data = Uri.fromFile(File("/sdcard/sample.jpg"))

Sending and Retrieving Data

2- Types Of Sending Data With Intents :

- **Extras**: one or more pieces of information as a collection of key-value pairs Or in a Bundle
  - **putExtra(String name, int value)**

    intent.putExtra("level", 406)
  - **putExtra(String name, String[] value)**

    val foodList= arrayOf("Rice", "Beans", "Fruit")

    intent.putExtra("food", foodList)
  - **putExtras(bundle)**

    if lots of data, first create a bundle and pass the bundle

Sending and Retrieving Data

Retrieving The Data

- **Get Data**
  - val laocateURI = intent.data
- **Get Extra (Integer)**
  - val level = getIntExtra(“Key” , DefualtValue)
- **Get All Data at Once As Bundle**
  - val bundle = intent.Extras

Save State

Save State

Users expect the UI state to remain the same after a configuration change (like a rotation) or if the app is terminated while in the background

- When an activity is destroyed and then restarted (Config change)—or if the app is terminated and later resumed (onPause)
- it’s important to store the necessary data so that the UI can be reconstructed to its previous state :
  - Use **Bundle** provided by **onSaveInstanceState().**
  - **onCreate()** receives the **Bundle** as an argument when activity is created again

Save State

System saves only:

➤ State of views with a unique ID (android: id) such as text entered into EditText.

➤ Intent that started activity and data in its extras

You are responsible for saving other activity and user progress data

Save State

Saving Instance State

Implement onSaveInstanceState() in your Activity

- Called by Android runtime when there is a possibility the Activity may be destroyed
- Saves data only for this instance of the Activity during the **current session**

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState) // Call the superclass

    // Add custom data to the outState Bundle
    outState.putString("key", value)
}
```

Restore State

OnCreate() :

- is called when the activity is **first created or recreated** (e.g., after a configuration change)
- This is the **primary place** to restore data because it is called in all cases where the activity is being initialized.
- It is ideal for **restoring critical data** needed to set up the activity (e.g., variables, or objects).

Restore State

OnCreate() :

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    if(savedInstanceState != null) {
        val myValue = savedInstanceState.getString("key")
        if(myValue != null) {
            // Continue your work task with myValue
        }
    }
}
```

Restore State

OnRestoreInstanceState() :

- is called after onStart() when the activity is being recreated.
- It is not called when the activity is created for the first time.
- This method is specifically designed for restoring the UI state after the activity's views have been initialized.
- It is useful when you need to restore data after the views are fully created (e.g., scroll position, selected item, etc.).

Restore State

OnRestoreInstanceState() :

```
override fun onRestoreInstanceState(savedInstanceState: Bundle) {
    super.onRestoreInstanceState(savedInstanceState)

    val value = savedInstanceState.getString( key: "Alshimaa")
    if(value != null){
        Log.d( tag: "TAG" ,value.toString() )
    }

}
```

Instance State and Restart The App

When U stop and Restart a new App Session :

- The activity Instance State are lost
- Your Activity will revert to their default Appearance.

IF U Need To save the User’s Data Between The App sessions :

- Use Shared Preference and Database .

Thank You