1. Resources Overview – Styles and Themes

Definition:

- Styles: A style in Android defines the look and feel for UI components such as buttons, text fields, etc. It is a collection of properties that can be applied to views in an XML layout file.

- Themes: A theme is a broader style that can be applied to an entire activity or application. It is essentially a collection of styles for different UI components like windows, text, buttons, etc., that are applied globally.

Steps to Create:

1. Create a new XML file under res/values/styles.xml.

2. Define a style with attributes like text color, background color, padding, etc.

3. You can create a theme by referencing styles for different elements (e.g., background, text color) for the entire app or activity.

Syntax Example for Style:

xml
Copy

```
<!-- res/values/styles.xml -->
<resources>
 <style name="CustomStyle">
 <item name="android:textColor">#FF0000</item>
 <item name="android:background">@color/white</item>
 </style>
</resources>
```

Syntax Example for Theme:

xml
Copy

```
<!-- res/values/styles.xml -->
```

```
<resources>
 <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">  <item
name="android:windowBackground">@drawable/background_image</item>  <item
name="android:textColor">#FFFFFF</item>
 </style>
</resources>
```

Steps to Apply a Theme:

1. Define a theme in res/values/styles.xml.
2. Apply the theme in AndroidManifest.xml under the <application> tag.

xml
Copy

```
<application
 android:theme="@style/AppTheme"
 android:name=".MyApplication">
 ...
</application>
```

Real-World Application:

- Styles and themes are crucial for creating a consistent design across all screens in an app. For example, applying a dark theme across the app for night mode.

2. Menu: Option Menu

Definition: The option menu is a menu that appears when the user presses the menu button on a device. It provides additional actions or options that are not part of the main UI.

Steps to Create:

1. Override onCreateOptionsMenu(Menu menu) in the Activity.
2. Inflate a menu resource (XML file) into the menu object.

3. Handle menu item clicks with onOptionsItemSelected(MenuItem item).

Syntax Example:

xml
Copy

```
<!-- res/menu/menu_main.xml -->
<menu xmlns:android="http://schemas.android.com/apk/res/android">
 <item android:id="@+id/action_settings"
 android:title="Settings"
 android:orderInCategory="100"
 android:showAsAction="never"/>
</menu>
```

java
Copy

```
// MainActivity.java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
 getMenuInflater().inflate(R.menu.menu_main, menu);
 return true;
 }

@Override
public boolean onOptionsItemSelected(MenuItem item) {
 if (item.getItemId() == R.id.action_settings) {
 // Open settings
 return true;
 }
 return super.onOptionsItemSelected(item);
 }
```

Real-World Application:

- Often used in apps like messaging or photo gallery apps, where the option menu provides access to settings or app options.

an Android application that demonstrates how to use the Option Menu to perform simple actions, such as displaying a toast message, clearing text, and exiting the app.

Steps to Create the Option Menu:

1. Set up a new Android project in Android Studio.
2. Create a simple UI with a TextView and a Button.
3. Implement an Option Menu to add actions like clearing the text and exiting the app.
4. Handle user interactions with the menu.

Step 1: Create the Layout (activity_main.xml)

We'll create a simple layout with a TextView and a Button to interact with.

xml

Copy

```
<!-- res/layout/activity_main.xml -->

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

 android:layout_width="match_parent"

 android:layout_height="match_parent"

 android:padding="16dp">



 <!-- TextView to display a message -->

 <TextView
 android:id="@+id/textViewMessage"

 android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:text="Hello, World!"
android:textSize="24sp"
android:layout_gravity="center"
android:padding="20dp" />

<!-- Button to change text in TextView -->
<Button
android:id="@+id/buttonChangeText"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Change Text"
android:layout_gravity="center"
android:padding="10dp" />

</LinearLayout>
```

Step 2: Implement the Option Menu in MainActivity.java

Now, let's add the logic to create and handle the Option Menu in the MainActivity.java file.

java

Copy

```
// MainActivity.java

package com.example.optionmenuexample;


import android.os.Bundle;

import android.view.Menu;

import android.view.MenuItem;

import android.widget.Button;

import android.widget.TextView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;


public class MainActivity extends AppCompatActivity {


 private TextView textViewMessage;

 private Button buttonChangeText;


 @Override

 protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

 setContentView(R.layout.activity_main);
 // Initialize the TextView and Button
```

```
textViewMessage = findViewById(R.id.textViewMessage);

buttonChangeText = findViewById(R.id.buttonChangeText);


// Button click listener to change the text in TextView

buttonChangeText.setOnClickListener(v -> textViewMessage.setText("Text Changed!"));  }


// Create the Option Menu

@Override

public boolean onCreateOptionsMenu(Menu menu) {

getMenuInflater().inflate(R.menu.option_menu, menu);

return true;

}


// Handle item selection from the Option Menu

@Override

public boolean onOptionsItemSelected(MenuItem item) {

switch (item.getItemId()) {

case R.id.action_clear:

// Clear the text when the Clear option is selected

textViewMessage.setText("");
return true;
```

```
case R.id.action_exit:

// Exit the app when the Exit option is selected

finish();

return true;

default:

return super.onOptionsItemSelected(item);

}

}

}
```

Step 3: Create the Option Menu XML (res/menu/option_menu.xml)

Now, we need to define the option menu items in an XML file. This will include actions like Clear and Exit.

xml

Copy

```
<!-- res/menu/option_menu.xml -->

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item

 android:id="@+id/action_clear"

 android:title="Clear"

 android:icon="@android:drawable/ic_menu_delete"
 android:showAsAction="ifRoom"/>
```

```
<item

android:id="@+id/action_exit"

android:title="Exit"

android:icon="@android:drawable/ic_menu_close_clear_cancel"

android:showAsAction="ifRoom"/>

</menu>
```

Explanation:

- onCreateOptionsMenu(): This method is responsible for inflating the option menu from the XML file. When you press the menu button on the device, this menu will appear.
- onOptionsItemSelected(): This method handles the item selection from the menu. Based on the selected item, we can perform actions like clearing the text or exiting the app.

Step 4: Run the Application

1. When you run this app, the main screen will display a TextView with the text "Hello, World!" and a button labeled "Change Text."
2. When you click the "Change Text" button, the TextView will update to "Text Changed!"
3. Tap the menu button (three vertical dots) in the action bar, and you'll see options for Clear and Exit.
    - Clear will clear the text in the TextView.
    - Exit will close the application.

3. Context Menu

Definition: Context menus are menus that appear when the user long-presses on a UI element.

They provide contextual actions based on the selected item.

Steps to Create:

1. Register a view to show the context menu using registerForContextMenu(View view).
2. Override onCreateContextMenu() to inflate the menu.
3. Handle item selection in onContextItemSelected().

Syntax Example:

java
Copy

```
// MainActivity.java
@Override
protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
 registerForContextMenu(findViewById(R.id.my_view));
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo) {
 super.onCreateContextMenu(menu, v, menuInfo);
 getMenuInflater().inflate(R.menu.context_menu, menu);
}

@Override
public boolean onContextItemSelected(MenuItem item) {
 if (item.getItemId() == R.id.edit) {
 // Handle edit
 return true;
 }
 return super.onContextItemSelected(item);
}
```

Real-World Application:

- Used in apps like email or file managers, where users can long press an item to show a context menu with actions like delete, edit, or share.

context menu items to "Change Color" or "Clear Text" using item selection.

Step 1: Create the Layout (activity_main.xml)

Let's create a simple layout with a TextView that the user can long-press to see the context menu.

xml

Copy

```
<!-- res/layout/activity_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:padding="16dp">

 <!-- TextView to demonstrate the Context Menu -->
 <TextView
 android:id="@+id/textView"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Long press me to show context menu"
```

```
 android:textSize="18sp"
 android:padding="20dp"

 android:background="#EFEFEF"

 android:gravity="center" />


</LinearLayout>
```

Step 2: Implement the Context Menu in MainActivity.java

Now, let's implement the logic to handle the Context Menu in MainActivity.java.

java

Copy

```
// MainActivity.java

package com.example.contextmenuexample;


import android.graphics.Color;

import android.os.Bundle;

import android.view.ContextMenu;

import android.view.MenuItem;

import android.widget.TextView;

import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;


public class MainActivity extends AppCompatActivity {
```

```
private TextView textView;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

// Initialize the TextView
textView = findViewById(R.id.textView);

// Register the TextView for the context menu
registerForContextMenu(textView);
}

// Create the Context Menu
@Override
public void onCreateContextMenu(ContextMenu menu, android.view.View v,
ContextMenu.ContextMenuInfo menuInfo) {
super.onCreateContextMenu(menu, v, menuInfo);

// Inflate the context menu from a menu resource file
getMenuInflater().inflate(R.menu.context_menu, menu);  }
// Handle Context Menu item selection
```

```
@Override
public boolean onContextItemSelected(MenuItem item) {
switch (item.getItemId()) {
case R.id.action_change_color:
// Change the text color of the TextView
textView.setTextColor(Color.RED);
Toast.makeText(this, "Text color changed!", Toast.LENGTH_SHORT).show();  return
true;
case R.id.action_clear_text:
// Clear the text of the TextView
textView.setText("");
Toast.makeText(this, "Text cleared!", Toast.LENGTH_SHORT).show();  return
true;
default:
return super.onContextItemSelected(item);
}
}
}
```

Step 3: Create the Context Menu XML (res/menu/context_menu.xml)

Next, we need to define the context menu items in an XML file. This menu will appear when the user long-presses the TextView.

xml
Copy

```
<!-- res/menu/context_menu.xml -->

<menu xmlns:android="http://schemas.android.com/apk/res/android">

<item

 android:id="@+id/action_change_color"

 android:title="Change Color"

 android:icon="@android:drawable/ic_menu_edit"

 android:showAsAction="ifRoom" />


 <item

 android:id="@+id/action_clear_text"

 android:title="Clear Text"

 android:icon="@android:drawable/ic_menu_delete"

 android:showAsAction="ifRoom" />

</menu>
```

Explanation:

- registerForContextMenu(): This method registers the TextView for the context menu. When the user long-presses the TextView, the context menu will appear.

- onCreateContextMenu(): This method is called when the context menu is created. We inflate the context_menu.xml file to define the items in the context menu.

- onContextItemSelected(): This method handles the selection of a menu item. We have two options:

1. Change Color: Changes the color of the TextView text to red.

2. Clear Text: Clears the text from the TextView.

Step 4: Run the Application

1. Long press the TextView that says "Long press me to show context menu". 2.

The context menu will appear with two options: Change Color and Clear Text. 3.

Selecting Change Color will change the text color to red.

4. Selecting Clear Text will remove the text in the TextView

4. Notification

Definition: A notification is a message that can be displayed outside the normal UI of an app, usually in the notification shade, to alert users about events or updates.

Steps to Create:

1. Create a Notification object.

2. Use NotificationManager to post the notification.

Syntax Example:

java
Copy

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, "my_channel_id")
.setSmallIcon(R.drawable.notification_icon)
 .setContentTitle("Sample Notification")
 .setContentText("Hello, World!")
 .setPriority(NotificationCompat.PRIORITY_DEFAULT);

NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);
```

```
notificationManager.notify(1, builder.build());
```

Real-World Application:

- Used for messaging apps, news apps, or email clients to notify users of new messages or events.

1.

5. Broadcast Receivers

Definition: Broadcast receivers are components that listen for and respond to broadcast messages from other applications or the system (e.g., network changes, battery low).

Steps to Create:

1. Create a BroadcastReceiver class.
2. Register the receiver either statically (in AndroidManifest.xml) or dynamically (at runtime).

Syntax Example:

java
Copy

```
public class MyBroadcastReceiver extends BroadcastReceiver {
 @Override
 public void onReceive(Context context, Intent intent) {
 // Handle broadcast message
 }
}
```

xml
Copy

```
<!-- Manifest file -->
<receiver android:name=".MyBroadcastReceiver" android:enabled="true">
<intent-filter>
 <action android:name="android.intent.action.BOOT_COMPLETED"/>
```

```
</intent-filter>
</receiver>
```

Real-World Application:

- Used in apps for monitoring system events, like battery status or network changes, or to listen to custom events.

Example Application using Both Notification and Broadcast Receivers

To create a notification in Android that shows every 10 minutes with options to Dismiss and Snooze, we can follow these steps:

Overview:

- Notification will appear every 10 minutes.
- It will have two actions: Dismiss and Snooze.
    - Dismiss will remove the notification.
    - Snooze will dismiss the notification and show it again after a specific time (e.g., 5 minutes).

Steps:

1. Create a simple UI to trigger notifications.
2. Use AlarmManager to schedule notifications every 10 minutes.
3. Create a notification with actions (Dismiss and Snooze).
4. Handle the actions for dismissing and snoozing notifications.

Step 1: Create the Layout (activity_main.xml)

Create a simple layout with a button to simulate user-triggered notifications.

xml

Copy

```
<!-- res/layout/activity_main.xml -->
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:padding="16dp">

 <!-- Button to manually trigger a notification -->
 <Button
 android:id="@+id/buttonStartNotification"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:text="Start Notification"
 android:layout_gravity="center" />

</LinearLayout>
```

Step 2: Implement the Notification Logic (MainActivity.java)

We'll implement logic in MainActivity.java to trigger notifications every 10 minutes using AlarmManager, and add actions for Dismiss and Snooze.

java

Copy

```
// MainActivity.java

package com.example.notificationexample;


import android.app.AlarmManager;

import android.app.Notification;

import android.app.NotificationChannel;

import android.app.NotificationManager;

import android.app.PendingIntent;

import android.app.Service;

import

android.content.BroadcastReceiver;

import android.content.Context;

import android.content.Intent;

import android.os.Build;

import android.os.Bundle;

import android.widget.Button;

import android.widget.Toast;


import androidx.appcompat.app.AppCompatActivity;

import androidx.core.app.NotificationCompat;
```

```
import java.util.Calendar;


public class MainActivity extends AppCompatActivity {
 private static final int NOTIFICATION_ID = 1;

 private static final String CHANNEL_ID = "notification_channel";

private static final int ALARM_REQUEST_CODE = 100;


 @Override

 protected void onCreate(Bundle savedInstanceState) {

 super.onCreate(savedInstanceState);

 setContentView(R.layout.activity_main);


 Button buttonStartNotification = findViewById(R.id.buttonStartNotification);


 // Create the notification channel for devices with Android O and above  if

(Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {

NotificationChannel channel = new NotificationChannel(  CHANNEL_ID,

 "Notification Channel",

 NotificationManager.IMPORTANCE_DEFAULT);

channel.setDescription("This is a notification channel.");

 NotificationManager notificationManager =
getSystemService(NotificationManager.class);
```

```
notificationManager.createNotificationChannel(channel);  }
buttonStartNotification.setOnClickListener(v -> scheduleNotification());  }

// Method to schedule a notification every 10 minutes
private void scheduleNotification() {
AlarmManager alarmManager = (AlarmManager)
getSystemService(Context.ALARM_SERVICE);
Intent intent = new Intent(this, NotificationReceiver.class);

// PendingIntent to trigger the notification every 10 minutes
PendingIntent pendingIntent = PendingIntent.getBroadcast(
this,
ALARM_REQUEST_CODE,
intent,
PendingIntent.FLAG_UPDATE_CURRENT
);

// Set the alarm to trigger the notification every 10 minutes  if
(alarmManager != null) {
Calendar calendar = Calendar.getInstance();
calendar.setTimeInMillis(System.currentTimeMillis());
calendar.add(Calendar.MINUTE, 10); // Start in 10 minutes
```

```
alarmManager.setRepeating(
AlarmManager.RTC_WAKEUP,

calendar.getTimeInMillis(),

AlarmManager.INTERVAL_HALF_HOUR, // Every 10 minutes

pendingIntent

);


Toast.makeText(this, "Notification Scheduled Every 10 Minutes",
Toast.LENGTH_SHORT).show();

}

}


// BroadcastReceiver to handle the actual notification

public static class NotificationReceiver extends BroadcastReceiver {


@Override

public void onReceive(Context context, Intent intent) {

// Create the notification

Notification notification = new NotificationCompat.Builder(context, CHANNEL_ID)

.setContentTitle("Reminder")

.setContentText("It's time for your reminder!")

.setSmallIcon(android.R.drawable.ic_dialog_info)

.setAutoCancel(true)
```

```
 .addAction(android.R.drawable.ic_menu_close_clear_cancel, "Dismiss",
getDismissPendingIntent(context))
 .addAction(android.R.drawable.ic_lock_idle_alarm, "Snooze",
getSnoozePendingIntent(context))

 .build();


 // Send the notification

 NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);

 if (notificationManager != null) {

 notificationManager.notify(NOTIFICATION_ID, notification);  }

 }


 // PendingIntent for Dismiss action

 private static PendingIntent getDismissPendingIntent(Context context) {

Intent dismissIntent = new Intent(context, DismissReceiver.class);

 return PendingIntent.getBroadcast(context, 0, dismissIntent,
PendingIntent.FLAG_UPDATE_CURRENT);

 }


 // PendingIntent for Snooze action

 private static PendingIntent getSnoozePendingIntent(Context context) {

Intent snoozeIntent = new Intent(context, SnoozeReceiver.class);

 return PendingIntent.getBroadcast(context, 0, snoozeIntent,
PendingIntent.FLAG_UPDATE_CURRENT);
```

```
}
}


// DismissReceiver to handle dismiss action

public static class DismissReceiver extends BroadcastReceiver {


@Override

public void onReceive(Context context, Intent intent) {

// Cancel the ongoing notification when Dismiss is clicked

NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);

if (notificationManager != null) {

notificationManager.cancel(NOTIFICATION_ID);

}

Toast.makeText(context, "Notification Dismissed", Toast.LENGTH_SHORT).show();  }

}


// SnoozeReceiver to handle snooze action

public static class SnoozeReceiver extends BroadcastReceiver {


@Override

public void onReceive(Context context, Intent intent) {

// Schedule the notification to appear again after 5 minutes (snooze)
AlarmManager alarmManager = (AlarmManager)
```

```
context.getSystemService(Context.ALARM_SERVICE);
 Intent snoozeIntent = new Intent(context, NotificationReceiver.class);
PendingIntent pendingIntent = PendingIntent.getBroadcast(  context,
 ALARM_REQUEST_CODE,
 snoozeIntent,
 PendingIntent.FLAG_UPDATE_CURRENT
 );

 if (alarmManager != null) {
 Calendar calendar = Calendar.getInstance();
 calendar.add(Calendar.MINUTE, 5); // Snooze for 5 minutes

 alarmManager.set(AlarmManager.RTC_WAKEUP,
calendar.getTimeInMillis(),
 pendingIntent);
 }

 // Cancel the current notification
 NotificationManager notificationManager = (NotificationManager)
context.getSystemService(Context.NOTIFICATION_SERVICE);
 if (notificationManager != null) {
 notificationManager.cancel(NOTIFICATION_ID);
 }
```

```
Toast.makeText(context, "Notification Snoozed for 5 Minutes",
Toast.LENGTH_SHORT).show();

}

}

}
```

Explanation:

1. Scheduling Notifications:
    - We use AlarmManager to schedule the notification to appear every 10 minutes.
    - PendingIntent is used to trigger the notification after a specified delay. 2. Notification with Actions:
    - We add two actions to the notification:
        - Dismiss: Cancels the notification immediately.
        - Snooze: Schedules the notification to appear again after 5 minutes.
3. Receivers:
    - NotificationReceiver: This receiver is triggered to show the notification.
    - DismissReceiver: This receiver handles the Dismiss action and cancels the notification.
    - SnoozeReceiver: This receiver handles the Snooze action and reschedules the notification after 5 minutes.
4. Notification Channel:

- A notification channel is required for Android 8.0 and above. It is used to group and categorize notifications.

Step 3: Update AndroidManifest.xml

You need to declare the receivers in the AndroidManifest.xml.

xml

Copy

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.notificationexample">

 <application
 android:allowBackup="true"
 android:icon="@mipmap/ic_launcher"
 android:label="Notification Example"
 android:theme="@style/Theme.AppCompat.Light.NoActionBar">

 <activity android:name=".MainActivity">
 <intent-filter>
 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
 </activity>
 <!-- Declare the receivers -->
```

```
<receiver android:name=".NotificationReceiver" />

<receiver android:name=".DismissReceiver" />

<receiver android:name=".SnoozeReceiver" />

</application>


</manifest>
```

Step 4: Run the Application

2. When you press the Start Notification button, a notification will be scheduled to show every 10 minutes.

3. You will see the notification with Dismiss and Snooze actions.

4. Dismiss will remove the notification.

5. Snooze will dismiss the notification and show it again after 5 minutes.

6. Web View

Definition: A WebView is a view that displays web pages within an app, allowing for embedded browsing functionality.

Steps to Create:

1. Add a WebView to the layout file.

2. Use WebView in Java code to load a URL.

Syntax Example:

xml

Copy

```
<WebView
 android:id="@+id/webview"
 android:layout_width="match_parent"
 android:layout_height="match_parent"/>
```

java

Copy

```
WebView webView = findViewById(R.id.webview);
webView.loadUrl("https://www.example.com");
```

Real-World Application:

- Common in apps that need to display web content, like news apps, social media apps, or e-commerce apps with embedded product pages.

WebView in Android with External and Internal Link Handling

A WebView in Android allows you to display web pages inside your application. It's useful for showing content from the web, such as HTML, JavaScript, and external links, without needing to open a browser.

In this example, we will:

- Display a webpage in a WebView.
- Handle external links (links that open in an external browser).
- Handle internal links (links that open within the WebView).

Steps:

1. Create a simple UI with a WebView.
2. Load a web page in the WebView.
3. Handle external links by opening them in an external browser.
4. Handle internal links to load them within the WebView.

Step 1: Create the Layout (activity_main.xml)

We will create a simple layout with a WebView that will display the web page.

xml

Copy

```
<!-- res/layout/activity_main.xml -->

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

 android:layout_width="match_parent"

 android:layout_height="match_parent"

 android:padding="16dp">


 <!-- WebView to display web pages -->

 <WebView

 android:id="@+id/webview"

 android:layout_width="match_parent"

 android:layout_height="match_parent" />

</LinearLayout>
```

Step 2: Implement WebView in MainActivity.java

Now, we will add logic in MainActivity.java to load a web page into the WebView, handle external links (by opening them in a browser), and handle internal links (by loading them inside the WebView).

java

Copy

```
// MainActivity.java

package com.example.webviewexample;


import android.content.Intent;

import android.net.Uri;

import android.os.Bundle;

import android.webkit.WebChromeClient;

import android.webkit.WebResourceRequest;

import android.webkit.WebView;

import android.webkit.WebViewClient;

import androidx.appcompat.app.AppCompatActivity;


public class MainActivity extends AppCompatActivity {


 private WebView webView;


 @Override

 protected void onCreate(Bundle savedInstanceState) {

 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);
```

```
// Initialize the WebView
webView = findViewById(R.id.webview);

// Enable JavaScript for the WebView
webView.getSettings().setJavaScriptEnabled(true);

// Set up a WebViewClient to handle internal links
webView.setWebViewClient(new WebViewClient() {
@Override
public boolean shouldOverrideUrlLoading(WebView view, WebResourceRequest
request) {
// If the link is external, open it in the default browser
String url = request.getUrl().toString();
if (url.startsWith("http://") || url.startsWith("https://")) {
return false; // Load it in the WebView itself
} else {
// Handle external links (e.g., phone numbers or other schemes)  Intent intent
= new Intent(Intent.ACTION_VIEW, Uri.parse(url));  startActivity(intent);
return true; // Prevent WebView from loading the link
}
}
});
```

```
// Set up a WebChromeClient for displaying the progress of page loading (optional)
webView.setWebChromeClient(new WebChromeClient());

// Load a URL in the WebView (for example, Google's homepage)
webView.loadUrl("https://www.example.com");
}

@Override
public void onBackPressed() {
// Handle the back button to go back in the WebView history
if (webView.canGoBack()) {
webView.goBack();
} else {
super.onBackPressed();
}
}
}
```

Step 3: Handle External and Internal Links

1. Internal Links:
    - When a user clicks on a link within the WebView (e.g., links that start with http:// or https://), the WebView will load those links within the same WebView.

2. External Links:

    - If the WebView encounters a link that should not be loaded inside the WebView (such as a mailto: link or a link to another application), we open it in the default browser by creating an Intent.

Step 4: Update AndroidManifest.xml

Make sure you have the appropriate permissions in your AndroidManifest.xml to access the internet:

xml

Copy

```
<!-- AndroidManifest.xml -->

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="com.example.webviewexample">


 <application

 android:allowBackup="true"

 android:icon="@mipmap/ic_launcher"

 android:label="WebView Example"

 android:theme="@style/Theme.AppCompat.Light.NoActionBar">


 <activity android:name=".MainActivity">

 <intent-filter>

 <action android:name="android.intent.action.MAIN" />
 <category android:name="android.intent.category.LAUNCHER" />
```

```
</intent-filter>

 </activity>


 <!-- Permission to access the internet -->

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


</application>


</manifest>
```

Explanation of Code:

1. WebView Setup:
    - webView.getSettings().setJavaScriptEnabled(true); enables JavaScript for the WebView, allowing interactive web pages to work properly.
    - webView.setWebViewClient(new WebViewClient()) is used to handle the internal links, preventing them from opening in an external browser.
2. Handling Internal and External Links:
    - shouldOverrideUrlLoading() method is overridden to handle the link clicks. Internal links (those that start with http:// or https://) are loaded in the WebView. For other links, we use Intent to open them in an external browser or other applications.
3. WebChromeClient:
    - This is optional but can be used to handle tasks like loading progress indicators or custom actions related to the web content.
4. Back Button Handling:

- The back button is overridden to navigate back in the WebView's history, allowing users to go back to previously visited pages.

Step 5: Run the Application

1. When you run the app, the WebView will load the URL https://www.example.com.
2. If you click a link within the WebView that is an internal link (e.g., another webpage), it will load within the WebView.
3. If you click an external link (e.g., a phone number or a non-HTTP URL), it will open in the external browser or corresponding app.

7. Phone Call

Definition: A phone call can be initiated by an app using the Intent system, which triggers the phone's dialer.

Steps to Create:

1. Use an Intent to initiate a call.
2. The phone's dialer opens, and the user can call the number.

Syntax Example:

java
Copy
```
Intent callIntent = new Intent(Intent.ACTION_DIAL);
callIntent.setData(Uri.parse("tel:1234567890"));
startActivity(callIntent);
```
Real-World Application:

- Used in contact or messaging apps to initiate a phone call to a contact.

8. Bundle

Definition: A Bundle is used to pass data between activities, or between an activity and a fragment.

Steps to Create:

1. Create a Bundle and put data into it.

2. Pass it to another activity using Intent.

Syntax Example:

java
Copy

```
Bundle bundle = new Bundle();
bundle.putString("key", "value");

Intent intent = new Intent(this, SecondActivity.class);
intent.putExtras(bundle);
startActivity(intent);
```

Real-World Application:

- Passing data (e.g., user preferences or form data) between activities in apps.

9. SQLite: CRUD Operations (Create, Insert, and View)

Definition: SQLite is a lightweight database used to store structured data in Android. CRUD operations represent the basic operations: Create, Read, Update, and Delete.
SQLite in Android: Notes and Overview

SQLite is a lightweight, embedded database that stores data in a local file on the device. It's the

default database engine used for storage in Android applications. SQLite provides a robust way to manage and store data, especially in mobile applications where using a full-fledged relational database would be inefficient or impractical.

Key Features of SQLite in Android:

- Local Database: Data is stored on the device's file system in a single file, making it ideal for mobile applications.
- SQL Syntax: SQLite supports a large subset of SQL standards, allowing developers to perform CRUD operations (Create, Read, Update, Delete) on data.
- Zero Configuration: SQLite doesn't require a server or network connection, making it simpler to set up and use.
- Lightweight: It has a small memory footprint and is optimized for mobile devices.

SQLite Basics in Android

1. SQLiteOpenHelper:
    - In Android, SQLite databases are managed using the SQLiteOpenHelper class.
    - This class simplifies database creation, version management, and version upgrades.
2. SQLiteDatabase:
    - Represents the database and provides methods for performing SQL operations like insert, update, delete, and query.
3. Cursor:
    - A Cursor is returned when querying the database. It is used to traverse the result set of a query.

Steps to Create:

1. Define a SQLiteOpenHelper class for database creation and version management.

2. Perform CRUD operations through the SQLiteDatabase object.

```
package com.example.sqliteexample;

import android.database.sqlite.SQLiteDatabase;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

        private EditText editTextNote;
        private Button buttonAdd;
        private SQLiteDatabase database;

        @Override
        protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 // Initialize UI elements
 editTextNote = findViewById(R.id.editTextNote);
 buttonAdd = findViewById(R.id.buttonAdd);

 // Open the database (if it doesn't exist, it will be created)
 database = openOrCreateDatabase("notes.db", MODE_PRIVATE, null);

 // Create the table if it doesn't exist
 String createTableQuery = "CREATE TABLE IF NOT EXISTS notes (" +  "id
INTEGER PRIMARY KEY AUTOINCREMENT, " +  "note TEXT)";
 database.execSQL(createTableQuery);

 // Button click to add a note
 buttonAdd.setOnClickListener(new View.OnClickListener() {
```

```
@Override
 public void onClick(View v) {
 String note = editTextNote.getText().toString().trim();  if
(!note.isEmpty()) {
 // Insert the note into the database
 long id = insertNote(note);

 // Show success message
 if (id != -1) {
 Toast.makeText(MainActivity.this, "Note added with ID: " + id,
Toast.LENGTH_SHORT).show();
 } else {
 Toast.makeText(MainActivity.this, "Failed to add note",
Toast.LENGTH_SHORT).show();
 }

 // Clear the input field
 editTextNote.setText("");
 } else {
 Toast.makeText(MainActivity.this, "Please enter a note",
Toast.LENGTH_SHORT).show();
 }
 }
 });
          }

          // Method to insert a new note into the database
          public long insertNote(String note) {
 String insertQuery = "INSERT INTO notes (note) VALUES ('" + note + "')";
database.execSQL(insertQuery);

 // Get the ID of the last inserted row
 Cursor cursor = database.rawQuery("SELECT last_insert_rowid()", null);  long
id = -1;
 if (cursor != null) {
 if (cursor.moveToFirst()) {
 id = cursor.getLong(0);
 }
 cursor.close();
```

```
}
return id;
        }

        @Override
        protected void onDestroy() {
super.onDestroy();
// Close the database when the activity is destroyed
if (database != null && database.isOpen()) {
database.close();
}
        }
}
```

Real-World Application:

- Used for storing user preferences, offline data, or simple database management in apps like note-taking apps or to-do list apps.